Case 5:03-cv-02289-JW Document 154 Filed 07/16/2004 Page 1 of 24

Exhibit A ('432 Patent)

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
13. A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising	During manufacture of a desired application specific integrated circuit (ASIC) chip that is designed to perform a specific purpose, a process of designing the desired ASIC using a computer, the process comprising: ("application specific integrated circuit (ASIC)"= an integrated circuit chip designed to perform a specific function.) Support '432 Patent: Column 1, lines 13-17; column 2, lines 15-20. RCL000207-223. "computer-aided design": The use of computers to aid in design layout and analysis. IEEE Standard Dictionary of Electrical and Electronic Terms,	A. "A computer-aided design process for designing" -- a process that uses a computer for designing, as distinguished from a computer-aided manufacturing process, which uses a computer to direct and control the manufacturing process. *Intrinsic Evidence:* **'432 patent:** 1:9-12; 16:34-65 [Attachment 2[1], DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 16-18[2]; **CAD Tool Integration For ASIC Design:** at 364-365 [Attachment 21, KBSC000031-KBSC000036]. *Dictionaries, Treatises, Textbooks, etc.:* **IBM Dictionary of Computing:** at 129-130 [Attachment 16, DEF083932-DEF084703]; **IC Mask Design:** at 1-24 Attachment 17,

[1] Referenced attachments are attached to this Exhibit A.

[2] "Decl. Kowalski ¶ ¶" refers to the pertinent paragraphs in the Declaration and Summary of Opinions of Dr. Thaddeus J. Kowalski on Construction of Disputed Claim Terms of United States Patent No. 4,922,432 that support Synopsys' and Defendants' proposed constructions and oppose Ricoh's proposed constructions for the disputed claim term, phrase or clause. Ricoh objects to the citation of extrinsic evidence by Synopsys and the Aeroflex defendants, and reserves its right to seek discovery regarding any such extrinsic evidence and respond with its own extrinsic evidence.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	Fourth Edition (1988). RCL011382-388 at RCL011384. "function": A specific purpose of an entity or its characteristic action. IEEE Standard Dictionary of Electrical and Electronic Terms, Fourth Edition (1988). RCL011382-388 at RCL011386. "function": Any of a group of related actions contributing to a larger action. Merriam-Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011400.	DEF085303-DEF085329]; **Microchip Fabrication:** at 76-82, 274-278 [Attachment 18, DEF085330-085343]; **IEEE Standard Dictionary of Electrical and Electronics Terms:** at 180 [Attachment 19, DEF085299-085302]; **Webster's Dictionary:** at 343, 725 [Attachment 20, DEF085290-DEF085298]. B. "application specific integrated circuit" -- an interconnected miniaturized electronic circuit on a single piece of semiconductor material designed to perform a specific function, as distinguished from standard, general purpose integrated circuits, such as microprocessors, memory chips, etc. *Intrinsic Evidence:* **'432 patent:** 1:13-17; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 19-20; **CAD Tool Integration For ASIC Design:** at 363 [Attachment 21, KBSC000031-KBSC000036].

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
storing a set of definitions of architecture independent actions and conditions;	Placing in computer memory a library of definitions of the different architecture independent actions and conditions that can be selected for use in the desired ASIC, where the architecture independent actions and conditions do not imply any structure or implementing technology. ("architecture independent action and condition"= functional or behavioral aspects of a portion of a circuit (or circuit segment) that does not imply any set architecture, structure or implementing technology.) <u>Support</u> '432 Patent: Figs. 1a, 1b, 1c, 4; column 2, lines 6-20; column 2, lines 27-34; column 3, line 49 to column 4, line 4; column 4, lines 5-19; column 5, lines 20-22; column 7, lines 24-50; column 8, lines 23-51; column 10, lines 10-12; column 13, lines 2-31.	C.[3] "actions and conditions"-- are the logical steps and decisions that are represented as rectangles and diamonds in the flowchart; collectively logical operations. *Intrinsic Evidence:* **'432 patent:** 2:24-27; 3:50-59; 4:15-19; 4:61-63; 6:3-14; 7:20-23; 8:47-51; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 9, 11; October 1989 Examiner Interview Summary; November 1989 Amendment at 6-7 [Attachment 3, DEF011820-DEF012110]. *Extrinsic Evidence:* **Decl. Kowalski** ¶ 21. *Dictionaries, Treatises, Textbooks, etc.:* **IBM Dictionary of Computing:** at 479 [Attachment 16, DEF083932-DEF084703].

[3] Synopsys and the Defendants disagree with Ricoh's definition of "storing" on this step and the next two steps as "placing in computer memory." Storing means placing on any storage device "that is accessible by the processor for the computer system." IBM Dictionary of Computing at 654 (Attachment 16, DEF083932-DEF084703).

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	"action": A thing done. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011391. "architecture": A unifying or coherent form or structure. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011393. "independent": Not dependent; not requiring or relying on something else. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011394. "condition": Something essential to the appearance or occurrence of something else. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011405A	D. "architecture independent"[4] -- not including (i.e., excluding) a register transfer level (RTL) description or any other description that is hardware architecture dependent. An RTL description consists of: 1) defining the inputs, outputs, and any registers of the proposed ASIC; and, 2) describing for a single clock cycle of the ASIC how the ASIC outputs and any registers are set according to the values of the ASIC inputs and the previous values of the registers; an RTL description defines any control needed for the ASIC. *Intrinsic Evidence:* **'432 patent file history:** April 1989 Amendment at 8-10, 13; November 1989 Amendment at 7 [Attachment 3, DEF011820-DEF012110]; **'435 patent:** Fig. 4; 4:26-32; 5:27-35 [Attachment 4, DEF012503-DEF012518]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 22-26; **Computer Aided VLSI Design Vol.1 No. 4:** at 388

[4] This phrase was not in the original patent application and therefore violates the prohibition of Section 132 of 35 U.S.C. against adding new matter. Moreover, it is indefinite and inadequately described in the '432 patent. The only description is in the '432 patent's file history, which is in the negative.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		[Attachment 22, KBSC001109-KBSC001131]. E. "a set of definitions of architecture independent actions and conditions" -- a set of named descriptions defining the functionality and arguments for the available logical steps and decisions that may be specified in the flowchart; and excluding a register transfer level description. *Intrinsic Evidence:* **'432 patent:** 4:61-63; 5:20-22; 6:3-14; 7:25-50; 8:47-51; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 27-28.
storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;	Placing in computer memory a library of cell information that describe hardware cells capable of performing the different architecture independent actions and conditions placed in the library of definitions. ("hardware cells"= previously designed circuit components or structure that have specific physical and functional characteristics used as building blocks for implementing an ASIC to be	F. "hardware cells" -- logic blocks for which the functional level (e.g., register transfer level), logic level (e.g., flip flop and gate level), circuit level (e.g., transistor level), and layout level (e.g., geometrical mask level) descriptions are all defined. *Intrinsic Evidence:* **'432 patent:** 2:34-39; 3:59-67; 5:15-20; 9:24-51; 16:34-65; 16:66-68 [Attachment 2, DEF012564-

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	manufactured.) Support '432 Patent: Fig. 4; column 2, lines 36-39; column 4, line 68 to column 5, line 3; column 5, lines 15-20; column 5, lines 22-25; column 9, lines 24-60; column 10, lines 10-12.	DEF012585]; **'016 patent:** 11:47-13:63; 14:3-22 [Attachment 5, DEF017265-DEF017284]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 29-30; **Computer Aided VLSI Design Vol.1 No. 4:** at 380 [Attachment 22, KBSC001109-KBSC001131]. G. "data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set" -- a set of named integrated circuit hardware cells that includes at least one hardware cell for each stored definition that may be specified for the available logical steps and decisions; where each named hardware cell has corresponding descriptions at the functional level (e.g., register transfer level), logic level (e.g., flip-flop and gate level), circuit level (e.g., transistor level), and layout level (e.g., geometrical mask level) that are all defined. *Intrinsic Evidence:* **'432 patent:** Fig. 4; 2:34-39; 3:59-67; 5:15-20; 5:23-25; 9:24-51; 16:34-65; 16:66-68 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:*

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Decl. Kowalski ¶¶ 31-32.
storing in an expert system knowledge base a set of rules for selecting hardware cells to perform the actions and conditions;	Placing in an expert system knowledge base, that uses a computer memory, a plurality of rules for selecting among the hardware cells placed in the hardware cell library, wherein the rules comprise the expert knowledge of highly skilled VLSI designers formulated as prescribed procedures. ("expert system knowledge base"= database used to store expert knowledge of highly skilled VLSI designers.) ("rules"= the expert knowledge of highly skilled VLSI designers formulated as prescribed procedures.) <u>Support</u> '432 Patent: Column 5, lines 6-8; column 8, line 65 to column 9, line 5; column 9, lines 14-20; column 10, lines 1-10; column 10, line 40 to column 11, line 14; column 11, lines 16-26; column 11, lines 30-32; column 11, line 46 to column 12, line 35. RCL000229-237.	H. "expert system" -- software executing on a computer system that attempts to embody the knowledge of a human expert in a particular field and then uses that knowledge to simulate the reasoning of such an expert to solve problems in that field. This system is comprised of a knowledge base containing rules, working memory containing the problem description, and an inference engine. It solves problems through the selective application of the rules in the knowledge base to the problem description, as distinguished from conventional software, which uses a predefined step-by-step procedure (algorithm) to solve problems. *Intrinsic evidence:* **'432 patent:** 2:58-63; 5:6-8; 8:58-60; 10:39-11:26; 14:50-59; 15:53-58; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent's file history:** April 1989 Amendment at 9-11, 15, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 7, 9 [Attachment 3, DEF011820-012110]; **'016 patent:** 2:65-3:8 [Attachment 5, DEF017265-DEF017284]; **'435 patent**: 7:32-9:35[Attachment 4, DEF012503-

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	"rule": A prescribed guide for conduct or action; an accepted procedure, custom or habit. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011405. "expert system": (1977): computer software that attempts to mimic the reasoning of a human specialist. Merriam-Webster's Collegiate Dictionary Tenth Edition (1999). RCL011408-410 at RCL011410.	DEF012518]; **'603 patent:** 1:44-49 [Attachment 6, DEF017456-DEF017482]; **An Overview of Logic Synthesis Systems:** at 170 [Attachment 7, DEF011962-DEF011968]; **The CMU Design Automation System:** at 75-77 [Attachment 8, DEF012045-DEF012052]. I. "Knowledge base" -- the portion of the expert system containing a set of rules embodying the expert knowledge for the particular field. *Intrinsic evidence:* **'432 patent:** 10:39-11:15; 14:50-59; 15:53-58; 16:34-65 [Attachment 2, DEF012864-DEF012585]; **'432 patent file history:** April 1989 Amendment at 9-11, 15, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 7, 9 [Attachment 3, DEF011820-DEF012110]; **'435 patent**: 7:32-9:35 [Attachment 4, DEF012503-DEF012518]. J. "a set of rules for selecting hardware cells to perform the actions and conditions" -- a set of rules, each having an antecedent portion (IF) and a consequent portion (THEN), embodying the knowledge of expert designers for application specific integrated circuits, which enables the

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		expert system to map the specified stored definitions for each logical step and decision represented in the flowchart to a corresponding stored hardware cell description. *Intrinsic evidence:* **'432 patent:** 2:58-63; 8:20-30; 8:58-9:62; 10:39-11:26; 14:50-59; 15:53-58; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 9-11, 15, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 7, 9 [Attachment 3, DEF011820-DEF012110]; **'435 patent**: 7:32-9:35 [Attachment 4, DEF012503-DEF012518]; **An Overview of Logic Synthesis Systems:** at 170 [Attachment 7, DEF011962-DEF011968]; **The CMU Design Automation System:** at 75-77 [Attachment 8, DEF012045-DEF012052]. *Extrinsic Evidence for all of the above phrases (H, I, & J) in this step:* **Decl. Kowalski** ¶¶ 33-45; **Computer Aided VLSI Design Vol.1 No. 4:** 351, 377-381, 383, 388-389 [Attachment 22, KBSC001109-KBSC001131]; **'669 patent:** Abstract, 1:13-16, 2:28-33, 4:31-62; 5:21-38, 5:58-68, 6:1-7:68, 17:62 [Attachment 23, DEF 080579-DEF080605]; **The VLSI**

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Design Automation Assistant: at 889-90 [Attachment 28, DEF018660-DEF018664]; **A Rule-Based Logic Circuit Synthesis System for CMOS Gate Arrays**: at 597 [Attachment 29, DEF018277-DEF018283]. *Dictionaries, Treatises, Textbooks, etc. for all of the above phrases (H, I, & J in this step):* **Understanding Expert Systems:** 7-10, 29-30, 40, 42, 74-78, 99-110 [Attachment 24, DEF079512-DEF079741]; **An Artificial Intelligence Approach To VLSI Design:** at 9-15 [Attachment 25, DEF078425-DEF078455]; **Artificial Intelligence Terminology:** at 6 [algorithm], 10 [antecedent], 53 [consequent], 86-87 [expert system], 127 [inference engine], 140 [knowledge based system], 204 [production system], 223 [rule base, rule-based system], 281 [working memory] [Attachment 26, DEF079212-DEF079511]; **Microsoft Press Computer Dictionary:** at 136 [expert system] [Attachment 27, DEF083323-DEF083325]; **IBM Dictionary of Computing:** 591 [rule interpreter, rule-based system] [Attachment 16, DEF083932-DEF084703]; **Expert Systems: A Non-Programmer's Guide:** 8-10, 16 [Attachment 30, DEF082264-DEF082528]; **Expert Systems:**

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Tools and Applications: at 269 [Attachment 31, DEF079985-DEF080283]; **Expert Systems: Principles and Case Studies:** at 11-12 [Attachment 32, DEF079753-DEF079984]; **Knowledge-Based Systems: The View in 1986:** at 16 [Attachment 33, DEF083251-DEF083284].
describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions;	A user describing an input specification containing the desired functions to be performed by the desired ASIC. Support '432 Patent: Column 1, line 13 to column 2, line 3, column 2, lines 6-20, column 2, lines 21-24; column 2, lines 27-34; column 6, lines 58-60; column 13, lines 32-35; column 14, lines 7-29. RCL000207-223, RCL000229-237. "describe": To represent or give an account in words <~ a picture>; to represent by a figure, model, or picture: delineate. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011398. "description": A descriptive statement or account.	K. "describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions" -- the designer represents a sequence of logical steps (rectangles) and decisions (diamonds), and the transitions (lines with arrows) between them in a flowchart format for a proposed application specific integrated circuit. *Intrinsic evidence:* **'432 patent:** Figs. 1a, 5, &7; 2:21-27; 3:20-22; 3:50-59; 4:5-22; 4:35-38; 7:12-23; 16:34-65 [Attachment 2, DEF012564-012585]; **'432 patent file history:** April 1989 Amendment at 9, 11; October 1989 Examiner Interview Summary; November 1989 Amendment at 6-7 [Attachment 3, DEF011820-DEF012110]; **'016 patent:** 7:33-9:52 [Attachment 5, DEF017265-DEF017284]; **'435 patent:** 4:26-33 [Attachment 4, DEF012503-DEF012518]; **Flamel: A High-**

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011398.	**Level Hardware Compiler:** 260-261 [Attachment 9, DEF012034-DEF012044]; **An Overview of Logic Synthesis Systems:** at 168-170 [Attachment 7, DEF011962-DEF011968]; **Methods Used in an Automatic Logic Design Generator (ALERT):** at 595 [Attachment 10, DEF011969-DEF011989]; **Quality of Designs From an Automatic Logic Generator (ALERT):** at 71 [Attachment 11, DEF012005-DEF012023]; **The CMU Design Automation System:** at 73-74 [Attachment 8, DEF012045-DEF012052]; **A New Look at Logic Synthesis:** at 544 [Attachment 12, DEF012024-DEF012030]; **Experiments in Logic Synthesis:** at 235 [Attachment 13, DEF011990-DEF011994]; **CAD Systems for IC Design:** at 3, 7 [Attachment 14, DEF011951-DEF011961]; **Verifying Compiled Silicon:** at 2 [Attachment 15, DEF011948-DEF011950]; **'442 patent:** 5:3-46, [Attachment 35, DEF012392-DEF012401; '603 patent: 2:41-61 [Attachment 6, DEF012392-DEF012401]. *Extrinsic Evidence:* **Decl. Kowalski ¶¶** 46-49; **Computer Aided VLSI Design Vol.1 No. 4:** 351, 377-381, 383, 388-389 [Attachment 22, KBSC001109-KBSC001131];

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Silicon Compilation: at 48-49 [Attachment 34, DEF076335-DEF076341]. *Dictionaries, Treatises, Textbooks, etc.:* **Webster's Ninth New Collegiate Dictionary** at 343, 1074, 1073 [Attachment 20, DEF085290-DEF085298].
specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and	Specifying for each desired function to be performed by the desired ASIC one of the definitions of the architecture independent actions and conditions stored in the library of definitions that is associated with the desired function. ("specifying"= mapping or associating a desired function to be performed by the manufactured ASIC with a definition from the library of definitions.) <u>Support</u> '432 Patent: Column 5, lines 20-22; column 7, lines 24-50; column 8, lines 23-51; column 8, lines 65-67; column 9, lines 8-18; column 10, lines 10-12; column 13, lines 2-31.	L. "specifying for each described action and condition of the series one of said stored definitions" -- the designer assigns one definition from the set of stored definitions for each of the described logical steps and decisions represented in the flowchart. *Intrinsic Evidence:* **'432 patent:** Fig. 5; 3:20-22; 4:61-63; 5:20-22; 7:24-25; 8:23-26; 8:51-56; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'016 patent:** 6:12-32 [Attachment 5, DEF017265-DEF017284]. *Dictionaries, Treatises, Textbooks, etc.* **Webster's Ninth New Collegiate Dictionary** at 1132 [Attachment 20, DEF085290-DEF085298]. M. "which corresponds to the desired action or condition to be performed" -- each specified definition must correspond to the intended step or

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		decision to be performed. *Intrinsic Evidence:* **'432 patent:** Fig. 5; 3:20-22; 4:61-63; 5:20-22; 7:24-25; 8:23-26; 8:51-56; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'016 patent:** 6:12-32 [Attachment 5, DEF017265-DEF017284]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 50-52. *Dictionaries, Treatises, Textbooks, etc.:* **Webster's Ninth New Collegiate Dictionary** at 110 [Attachment 20, DEF085290-DEF085298].
selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware	Selecting from the plurality of hardware cells in the hardware cell library a hardware cell for performing the desired function of the desired ASIC through application of the rules; and generating a netlist that identifies the hardware cells needed to perform the function of the desired ASIC and the necessary parameters for connecting the respective inputs and outputs of each hardware cell, the netlist is passed to the next subsequent step in the process for manufacturing the desired ASIC. ("netlist"= a description of the hardware components	Synopsys and Defendants provide their constructions for the disputed claim terms, phrases, and clauses and support for these two separate "selecting" and "generating for the selected integrated hardware cells" steps below and separately for each step. Synopsys and Defendants dispute Ricoh's attempt to improperly and misleadingly combine these two separate steps.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.	(and their interconnections) needed to manufacture the ASIC as used by subsequent processes, e.g., mask development, foundry, etc.) Support '432 Patent: Figs. 4, 9, 13; column 1, lines 17-26; column 2, lines 34-36; column 2, lines 42-44; column 5, lines 25-29; column 5, lines 35-40; column 8, lines 21-23; column 8, lines 26-41; column 8, lines 58-64; column 9, lines 8-24; column 9, line 64 to column 10, line 7; column 10, lines 13-34; column 13, line 36 to column 14, line 6. RCL000207-223, RCL000229-237. "expert system": (1977): computer software that attempts to mimic the reasoning of a human specialist. Merriam-Webster's Collegiate Dictionary Tenth Edition (1999). RCL011408-410 at RCL011410. "interconnection": To connect with one another. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011403.	

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
	"requirement": Something wanted or needed-Necessity. Merriam-Webster's Ninth New Collegiate Dictionary (1987). RCL011389-407 at RCL011404.	
selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert	See the "selecting" row at column 2, *supra*, for Ricoh's analysis of this claim element. Because the selecting step was amended to include (and was granted as including) both "applying" and "generating" substeps, Ricoh has interpreted this claim element as a single claim element. Ricoh believes that Synopsys and the ASIC Defendants' attempt to cull out the "generating" substep as a separate and distinct claim element is an improper rewriting of the claim.	N. "selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit" -- mapping the specified stored definitions for each logical step and decision represented in the flowchart to a corresponding stored hardware cell description. *Intrinsic Evidence:* **'432 patent:** Fig. 4; 3:16-19; 4:66-5:3; 5:22-29; 8:31-37; 8:58-60; 9:52-60; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 10 [Attachment 3, DEF011820-DEF012110]. *Extrinsic Evidence:* **Decl. Kowalski** ¶ 55. O. "said step of selecting a hardware cell comprising applying to the specified definition of the action

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
system knowledge base and		or condition to be performed, a set of cell selection rules stored in said expert system knowledge base" -- the mapping of the specified definitions to the stored hardware cell descriptions must be performed by an expert system having an inference engine for selectively applying a set of rules, each rule having an antecedent portion (IF) and a consequent portion (THEN), embodying the knowledge of expert designers for application specific integrated circuits, which enables the expert system to map the specified stored definitions for each logical step and decision represented in the flowchart to a corresponding stored hardware cell description. *Intrinsic Evidence:* **'432 patent:** Abstract; 2:58-63; 5:6-8; 8:29-37; 8:58-60; 9:8-13; 11:16-26; 16:34-65 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 8-11, 17; October 1989 Examiner Interview Summary; November 1989 Amendment at 4, 6-7, 9 [¶¶ Attachment 3, DEF011820-DEF012110]; **'435 patent**: 7:32-9:35[Attachment 4, DEF012503-DEF012518]; **'603 patent:** at 3:59-63 [Attachment 6, DEF017456-DF017482]; **'016 patent:** 3:5-8; 9:67-10:2, [Attachment 5,

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		DEF017265-DEF017284; **An Overview of Logic Synthesis Systems:** at 170 [Attachment 7, DEF011962-DEF011968]; **The CMU Design Automation System:** at 75-77 [Attachment 8 , DEF012045-DEF012052]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 56-57 Same as set forth for phrases H, I, & J above. *Dictionaries, Treatises, Textbooks, etc.* Same as set forth for phrases H, I & J above.

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.	See the "selecting" row at column 2, *supra*, for Ricoh's analysis of this claim element. Because the selecting step was amended to include (and was granted as including) both "applying" and "generating" substeps, Ricoh has interpreted this claim element as a single claim element. Ricoh believes that Synopsys and the ASIC Defendants' attempt to cull out the "generating" substep as a separate and distinct claim element is an improper rewriting of the claim.	P. "Netlist" -- a structural description that includes a custom controller type hardware cell and all other hardware cells required to implement the application specific integrated circuit's operations and any necessary interconnections including the necessary control and data path information for connecting the hardware cells and the controller. *Intrinsic Evidence:* **'432 patent:** Abstract; 1:17-37; 2:39-44; 4:39-43; 5:8-12; 5:30-40; 9:62-10:9; 12:31-35; 13:55-14:3; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 58-60. Q. "generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit" -- producing a list of the needed hardware cells by eliminating any mapped hardware cells that are redundant or otherwise unnecessary and producing a custom controller type hardware cell for providing the needed control for those other hardware cells and

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Intrinsic Evidence: **'432 patent:** Abstract; 1:17-37; 2:39-44; 4:39-43; 5:8-12; 5:30-40; 9:62-10:9; 13:55-14:3; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 61-62. R. "generating …interconnection requirements therefor" -- producing the necessary structural control paths and data paths for the needed hardware cells and the custom controller. *Intrinsic Evidence:* **'432 patent:** Abstract; Figs. 6 & 13-15; 1:17-37; 2:39-44; 3:23-25; 3:40-45; 4:39-43; 5:8-12; 5:30-40; 9:62-10:9; 13:55-14:3; 16:34-65 [Attachment 2, DEF012564-DEF012585]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 63-64.
14. A process as defined in claim 13, including generating from the netlist the mask data required to	The process of claim 13, including producing from the netlist of hardware cells to be included in the designed ASIC mask data which can be directly used by a chip foundry in the fabrication of the ASIC.	S. "generating from the netlist the mask data required to produce an integrated circuit having the desired function" -- producing, from the structural netlist, the detailed layout level geometrical information required for

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
produce an integrated circuit having the desired function.	<u>Support</u> '432 Patent: Figs. 1c, 2; column 1, lines 42-43; column 2, lines 44-49; column 3, line 68 to column 4, line 4; column 4, lines 44-46; column 5, lines 40-46.	manufacturing the set of photomasks that are used by the processes that directly manufacture the application specific integrated circuit. *Intrinsic Evidence:* **'432 patent:** Abstract; Fig. 1c; 1:42-44; 1:54-58; 2:44-49; 4:44-46; 5:40-46; 14:4-6; 16:34-68 [Attachment 2, DEF012564-DEF012585]; **'016 patent:** 1:41-47; 1:57-61; 4:11-13 [Attachment 5, DEF017265-DEF017284]. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 65-66.
15. A process as defined in claim 13 including the further step of generating data paths for the selected integrated circuit hardware cells.	The process of claim 13, including producing signal lines for carrying data to the hardware cells. <u>Support</u> '432 Patent: Figs. 11, 13; column 2, lines 39-40; column 3, lines 60-65. RCL000207-223.	T. "generating data paths for the selected integrated circuit hardware cells" -- producing the necessary structural descriptions of the data paths for the mapped hardware cells. *Intrinsic Evidence:* **'432 patent:** Abstract; 2:39-40; 4:63-66; 5:6-12; 5:30-37; 6:29-31; 6:37-43; 6:50-53; 9:62-10:9; 13:55-14:3; 16:34-65; 17:1-3 [Attachment 2, DEF012564-DEF012585]; **'016 patent:** 2:21-25 [Attachment 5, DEF017265-017284]. *Extrinsic Evidence:*

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Decl. Kowalski ¶¶ 67-69. *Dictionaries, Treatises, Textbooks, etc.:* **IEEE Standard Dictionary** at 898 [Attachment 19, DEF085344-DEF085347].
16. A process as defined in claim 15 wherein said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom.	The process of claim 15, wherein the step of producing signal lines for carrying data comprises applying rules, which are placed in computer memory, to produce the signal lines for carrying data to the hardware cells. Support '432 Patent: Column 4, lines 64-66. RCL000207-223.	U. "said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom" -- the generating step must be performed by at least an expert system having an inference engine for selectively applying a set of rules, each having an antecedent portion (IF) and a consequent portion (THEN), embodying the knowledge of expert designers for application specific integrated circuits, which enables the expert system to produce the necessary data paths for the mapped hardware cells. *Intrinsic Evidence:* **'432 patent:** Abstract; 5:6-12; 9:62-10:9; 13:55-14:3; 16:34-65; 17:1-7 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 11 [Attachment 3, DEF011820-DEF012110]; **'435 patent**: 7:32-9:35 [Attachment 4, DEF012503-DEF012518]; **An Overview of Logic Synthesis**

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Systems: at 170 [Attachment 7, DEF011962-DEF011968]; **The CMU Design Automation System:** at 75-77 [Attachment 8, DEF012045-DEF012052. *Extrinsic Evidence:* **Decl. Kowalski** ¶¶ 69-70. Same as set forth for phrases H, I, & J above. *Dictionaries, Treatises, Textbooks, etc.* Same as set forth for phrases H, I & J above.
17. A process as defined in claim 16 including the further step of generating control paths for the selected integrated circuit hardware cells.	The process of claim 16, including producing signal lines for carrying control signals to the hardware cells. Support '432 Patent: Figs. 11, 13; column 2, lines 40-42; column 3, lines 60-65. RCL000207-223.	V. "generating control paths for the selected integrated circuit hardware cells" --producing the necessary structural descriptions of the control paths for the selected hardware cells. *Intrinsic Evidence:* **'432 patent:** Abstract; Figs. 1b & 13-15; 1:17-37; 2:40-42; 3:59-65; 4:39-43; 4:63-65; 5:3-12; 5:30-36; 6:18-27; 11:49-51; 13:51-14:3; 16:34-65; 17:1-10 [Attachment 2, DEF012564-DEF012585]; **'432 patent file history:** April 1989 Amendment at 8 [Attachment 3, DEF011820-DEF012110]; **'016 patent:** 2:20-24 [Attachment 5, DEF017265-DEF017284].

Claim Element	Ricoh's Construction(s) And Support For Those Construction(s)	Synopsys' & Defendants' Construction(s) And Support For Those Construction(s)
		Extrinsic Evidence: **Decl. Kowalski** ¶¶ 71-72. *Dictionaries, Treatises, Textbooks, etc.:* **IEEE Standard Dictionary** at 898 [signal line], [Attachment 19, DEF085344-DEF085347].

THADDEUS J. KOWALSKI

73 Stone Ridge Road • Summit, NJ 07901 • +1 908 370 9317 • t.kowalski@spandt.com

Summary

Senior technical executive with extensive hands-on experience using internet, computing, and network technologies to solve global business problems. Exceptional business development and technology management. Worldwide, large-scale product and project management. Internationally patented, published, and recognized. Merger and acquisition evaluations and business turnarounds.

Software Practices and Technology

Summit, NJ **2003 – Present**

A limited liability company specializing in technical assessments and business turnarounds.

Owner

Controlled P&L and led strategy, marketing, staffing, and execution for a consulting service.

- Led a team of five elite CTO/CIO's in reviewing the Department of Homeland Security Science and Technology Directorate's enterprise architecture design.
- Developed a competitive analysis of wireless application interface platforms for New Venture Partners, which provided validation of novelty and utility to multiple wireless carriers.
- Planned and executed assessments for Thin Layer Partners, which steered them away from making a bad investment.
- Developed new features and patches for an open source multi-protocol instant messaging client, Gaim, which improved its portability and acceptance.

AT&T

Florham Park, Basking Ridge, Somerset, and Murray Hill, NJ **1977 – 2003**

$37 billion telecommunications company with over 50 million consumer and 4 million business customers.

Architecture and Strategy

Vice President and Chief Technology Officer, AT&T Labs, Advanced Real-Time Communications Architecture and Strategy (2002 – 2003)

Planned and created new domestic and global business services.

- Defined market and technical requirements for an automated call center service with natural language speech recognition, which launched a $1 billion potential revenue service (VoiceTone).
- Presented VoiceTone technology vision on over two dozen sales visits to top customers, which landed 8 contracts and created a high probability sales funnel of 12 customers.
- Conceptualized, funded, and demonstrated a voice-over-internet protocol (VoIP)-based portable office service, with $700 million potential revenue.
- Led the technical due diligence team on a $100 million merger and acquisition decision, which steered AT&T away from making the purchase.
- Created an international service plan for VoiceTone, fulfilling essential customer requirements.
- Planned and implemented the Voice-on-Net booth and the Global Network Operations Center VoIP showcase, which created 115 new customer relationships leading to $10 million in contracts.

THADDEUS J. KOWALSKI

73 Stone Ridge Road • Summit, NJ 07901 • +1 908 370 9317 • t.kowalski@spandt.com

AT&T (continued)

e*Business Strategy

Vice President and Chief Technology Officer, AT&T Business Services, e*AT&T (2000 – 2002)

*Defined internal and external e*commerce and e*business technology migration and architecture strategy for Business Services' customer self-service initiatives.*

- Created teams to systematically and efficiently evaluate, develop, and use newly formed business systems (such as e*CRM) and practices (such as XML, SOAP, and J2EE) during a $50 million business transformation, which established company-wide standards and licenses.
- Analyzed and improved reliability, availability, and security of the att.com corporate web site and a 4 million-customer business direct web site by 99.8%.
- Improved customer satisfaction for a vendor's on-line shopping service by 52%.
- Defined the market requirements, technical plan, architecture, and vendor requirements for a B2B gateway, which reduced operating costs by $1.5 million.

Software Consulting Services

Director, AT&T Labs, Software Practices and Technology (1997 – 2000)

Controlled P&L and led strategy, marketing, staffing, and planning for a 150-person internal consulting service interfacing with all levels of management and engineering.

- Transformed a disorganized $6 million organization into a $25 million consulting business that delivered more than 400 projects on time and within budget.
- Planned and executed 12 assessments of prospective acquisitions and partners such as Excite@Home, ICQ, and Tribal Voice for the senior leadership team, resulting in the restructuring of Excite@Home's business and partnerships with Tribal Voice and others.
- Planned and executed 28 assessments of internal projects with budgets exceeding $10 million for the senior leadership team, resulting in a successful turnaround rate of 90%.
- Managed the product realization of DjVu, the next-generation image processing technology used to publish scanned documents on the web, which resulted in a $5 million sale to LizardTech.
- Implemented a company-wide software quality award, which stimulated capability maturity model (CMM) level two and three certifications.
- Evaluated and demonstrated application service provider (ASP), content management, portal, and directory technology, which resulted in Concert Technologies's $5 million IP Futures program.

e*Commerce Development

Director, AT&T Business Markets Division, Intelligent Voice (1995 – 1997)

Transformed touch-tone interactive voice response (IVR) and find/follow-me services by incorporating research in automatic speech recognition (ASR) and text-to-speech synthesis (TTS).

- Created and managed AT&T's automatic speech recognition, electronic commerce, and intelligent agent services, which resulted in $15 million of revenue.
- Developed and deployed a lights-out 24x7 enterprise-managed network platform with billing, OA&M, speech recognition, and speech synthesis, which was 99.99% reliable.
- Established the Pipeline team of researchers and developers to bring cutting edge speech technology into products, which reduced costs by $5 million and decreased technology transfer time by 22%.

THADDEUS J. KOWALSKI

73 Stone Ridge Road • Summit, NJ 07901 • +1 908 370 9317 • t.kowalski@spandt.com

AT&T (continued)

Consumer Products Development

Director, AT&T Consumer Products, Intelligent Devices (1994 – 1995)

Designed innovative new products and services for consumers with a staff of over 275 people.

- Invented and created the HomeCenter Web-TV and 8222 screen phone hardware, software, and service bureaus in half the usual time. Product concept was sold for $430 million.
- Selected as the company's technology spokesperson for intelligent devices, which resulted in total print, TV, and radio coverage of 120 million, including Jay Leno's monolog on the Tonight Show.
- Managed the manufacturing cycle for HomeCenter Web-TV, 8222 screen phone, and the 8130 computer phone in the Americas, reducing defects by 28%.
- Chaired the Chairman's Software Leadership Group, which established company-wide process and software standards.

Systems and Software Research

Member of Technical Staff, Bell Labs, Systems and Software Research (1980 – 1994)

Invented, designed, implemented, and patented the VLSI design automation assistant, a seminal expert system that designed microcomputers; the C/C++ interpreter; the legacy systems asset recovery technique; a formal verification system for timed-system behavior; the PARASITE real-time operating system; and the monk text processing system.

UNIX Development

Member of Technical Staff, Bell Labs, Unix Support (1977 – 1980)

Designed, implemented, and maintained major subsystems of the UNIX operating system, the user commands, the file system check (FSCK) program, and the Equipment Test Package.

Education

Carnegie Mellon University, Pittsburgh, PA

- Doctor of Philosophy in Electrical Engineering (1984).
- Master of Science in Electrical and Computer Engineering (1978).

University of Michigan, Ann Arbor, MI

- Bachelor of Science in Computer Engineering (1977).

Awards

- Advanced Management Program for fast-track managers, AT&T Labs (1998).
- Two exceptional leadership awards, AT&T (1995 and 1997).
- Innovations '95, International Consumer Electronics Show (1995).
- Affirmative Action, Bell Labs (1993).
- Ten individual performance awards, AT&T (1987 – 1995).
- Best Presentation, International Design Automation Conference (1983 and 1985).

Patents and Publications

- Granted nine United States and five international patents, with over 20 pending.
- Authored two technical books.
- Authored almost one hundred journal articles.
- Presented over two hundred technical talks.

2

United States Patent [19]

Kobayashi et al.

[11] **Patent Number: 4,922,432**

[45] **Date of Patent: May 1, 1990**

[54] **KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS**

[75] Inventors: **Hideaki Kobayashi**, Columbia, S.C.; **Masahiro Shindo**, Osaka, Japan

[73] Assignees: **International Chip Corporation**, Columbia, S.C.; **Ricoh Company, Ltd.**, Tokyo, Japan

[21] Appl. No.: **143,821**

[22] Filed: **Jan. 13, 1988**

[51] **Int. Cl.**[5] **G06F 15/60**

[52] **U.S. Cl.** **364/490**; 364/489; 364/488; 364/521

[58] **Field of Search** 364/488–491, 364/521, 300, 513

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,635,208	1/1987	Coleby et al.	364/491
4,638,442	1/1987	Bryant et al.	364/489
4,648,044	3/1987	Hardy et al.	364/513
4,651,284	3/1987	Watanabe et al.	364/491
4,656,603	4/1987	Dunn	364/488
4,658,370	4/1987	Erman et al.	364/513
4,675,829	6/1987	Clemenson	364/513
4,700,317	10/1987	Watanabe et al.	364/521
4,703,435	10/1987	Darringer et al.	364/488
4,803,636	2/1989	Nishiyama et al.	364/491

FOREIGN PATENT DOCUMENTS

1445914	8/1976	United Kingdom	364/490

OTHER PUBLICATIONS

"Verifying Compiled Silicon", by E. K. cheng, VLSI Design, Oct. 1984, pp. 1–4.

"CAD System for IC Design", by M. E. Daniel et al., IEEE Trans. on Computer-Aided Design of Integrated Circuits & Systems, vol. CAD–1, No. 1, Jan. 1982, pp. 2–12.

"An Overview of Logic Synthesis System", by L. Trevillyan, 24th ACM/IEEE Design Automation Conference, 1978, pp. 166–172.

"Methods Used in an Automatic Logic Design Generator", by T. D. Friedman et al., IEEE Trans. on Computers, vol. C–18, No. 7, Jul. 1969, pp. 593–613.

"Experiments in Logic Synthesis", by J. A. Darringer, IEEE ICCC, 1980.

"A Front End Graphic Interface to First Silicon Compiler", by J. H. Nash, EDA 84, Mar. 1984.

"quality of Designs from An Automatic Logic Generator", by T. D. Friedman et al., IEEE 7th DA Conference, 1970, pp. 71–89.

"A New Look at Logic Synthesis", by J. A. Darringer et al., IEEE 17th D. A. Conference 1980, pp. 543–548.

Trevillyan—Trickey, H., Flamel: *A High Level Hardward Compiler*, IEEE Transactions On Computer Aided Design, Mar. 1987, pp. 259–269.

Parker et al., The *CMU Design Automation System—An Example of Automated Data Path Design*, Proceedings Of The 16th Design Automation Conference, Las Vegas, Nev., 1979, pp. 73–80.

An Engineering Approach to Digital Design, William I. Fletcher, Prentice–Hall, Inc., pp. 491–505.

Primary Examiner—Felix D. Gruber

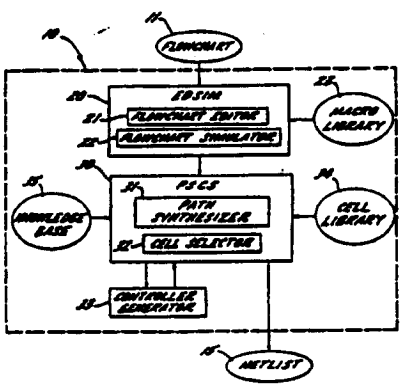
Assistant Examiner—V. N. Trans

Attorney, Agent, or Firm—Bell, Seltzer, Park & Gibson

[57] **ABSTRACT**

The present invention provides a computer-aided design system and method for designing an application specific integrated circuit which enables a user to define functional architecture independent specifications for the integrated circuit and which translates the functional architecture independent specifications into the detailed information needed for directly producing the integrated circuit. The functional architecture independent specifications of the desired integrated circuit can be defined at the functional architecture independent level in a flowchart format. From the flowchart, the system and method uses artificial intelligence and expert systems technology to generate a system controller, to select the necessary integrated circuit hardware cells needed to achieve the functional specifications, and to generate data and control paths for operation of the integrated circuit. This list of hardware cells and their interconnection requirements is set forth in a netlist. From the netlist it is possible using known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level topological information (mask data) required to produce the particular application specific integrated circuit.

20 Claims, 12 Drawing Sheets



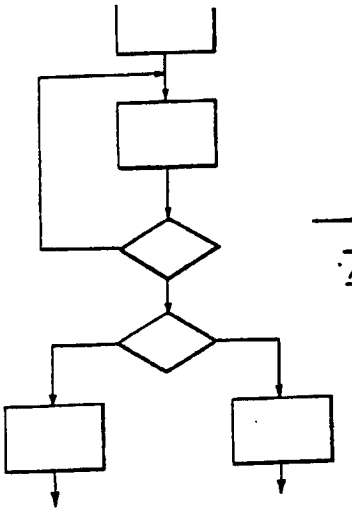


Fig. 1a.
FUNCTIONAL LEVEL

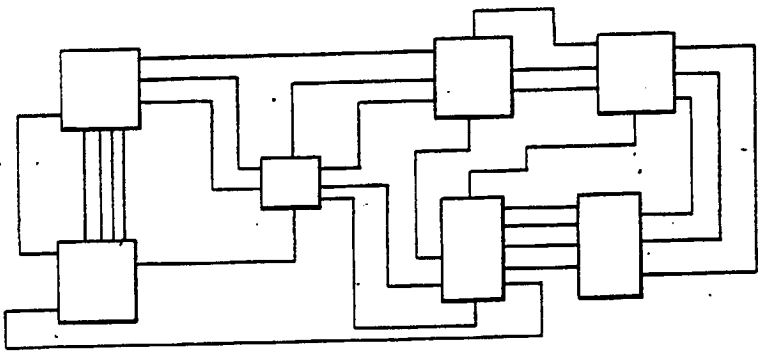


Fig. 1b.
STRUCTURAL LEVEL

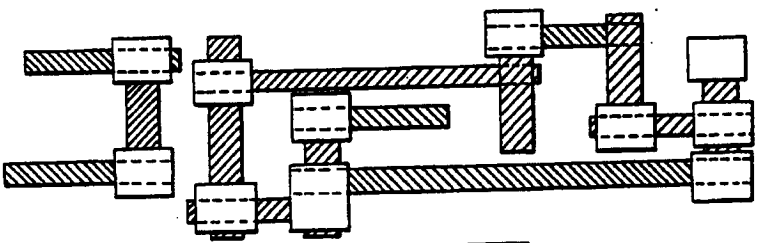


Fig. 1c.
PHYSICAL LAYOUT LEVEL

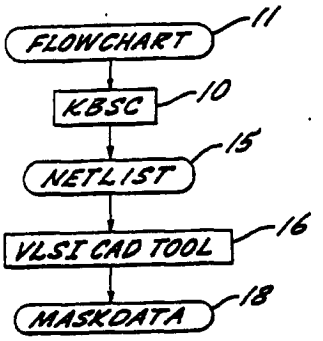

FLOWCHART
11
KBSC
10
NETLIST
15
VLSI CAD TOOL
16
MASKDATA
18

Fig. 2.

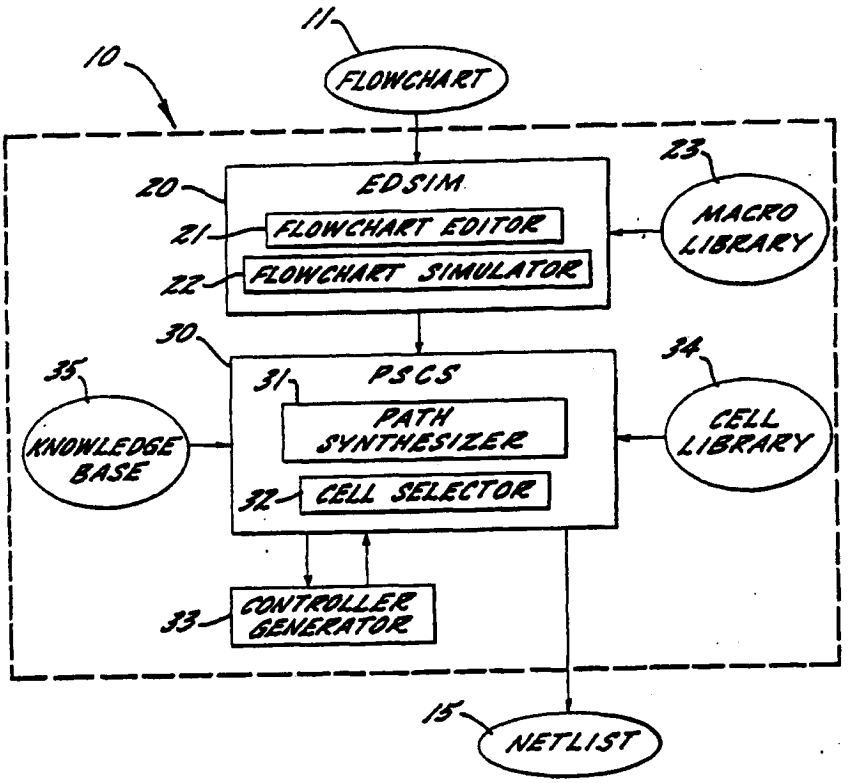

11
FLOWCHART
10
EDSIM
20
FLOWCHART EDITOR
21
FLOWCHART SIMULATOR
22
23
MACRO LIBRARY
30
PSCS
31
PATH SYNTHESIZER
CELL SELECTOR
32
35
KNOWLEDGE BASE
34
CELL LIBRARY
CONTROLLER GENERATOR
33
15
NETLIST

Fig. 3.

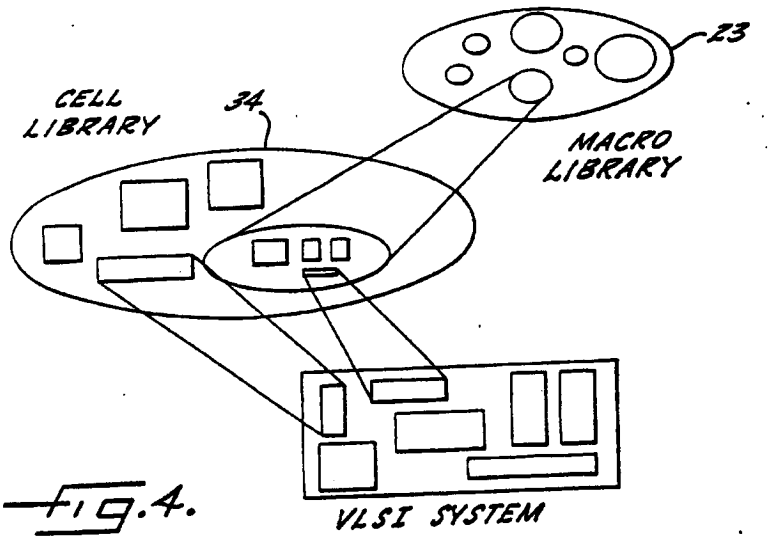

23
CELL LIBRARY
34
MACRO LIBRARY
VLSI SYSTEM

Fig. 4.

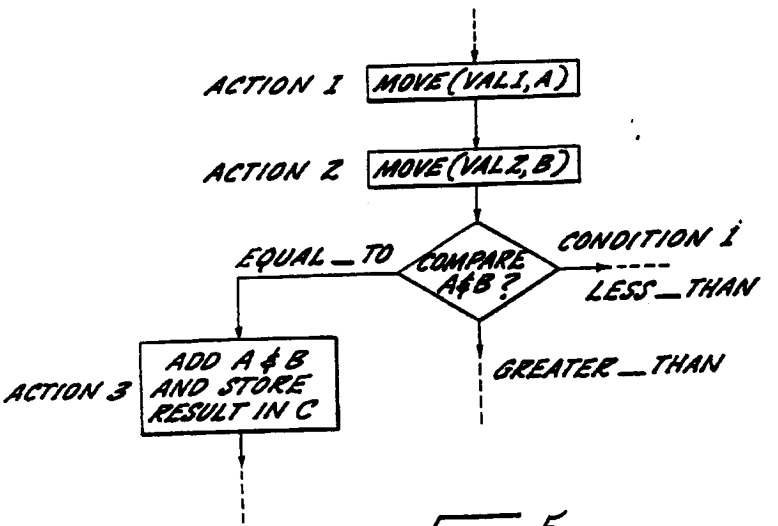

ACTION 1
MOVE(VAL1,A)
ACTION 2
MOVE(VAL2,B)
EQUAL_TO
COMPARE A&B ?
CONDITION 1
LESS_THAN
GREATER_THAN
ACTION 3
ADD A & B AND STORE RESULT IN C

Fig. 5.

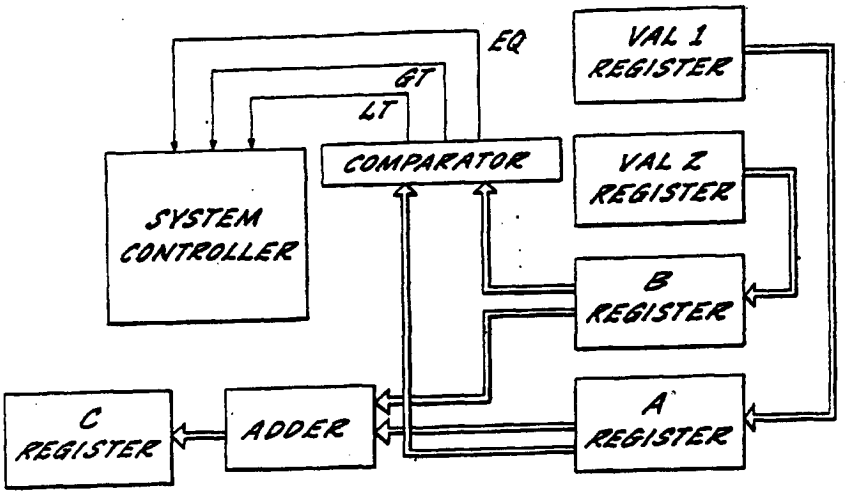

EQ
GT
LT
VAL 1 REGISTER
COMPARATOR
VAL 2 REGISTER
SYSTEM CONTROLLER
B REGISTER
C REGISTER
ADDER
A REGISTER

Fig. 6.

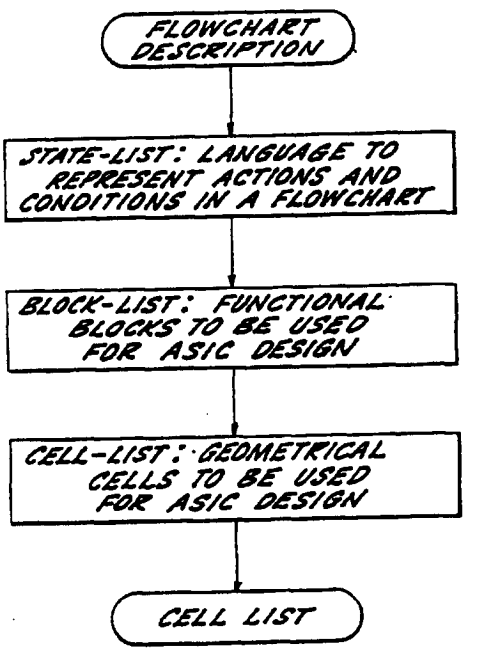

FLOWCHART DESCRIPTION
STATE-LIST: LANGUAGE TO REPRESENT ACTIONS AND CONDITIONS IN A FLOWCHART
BLOCK-LIST: FUNCTIONAL BLOCKS TO BE USED FOR ASIC DESIGN
CELL-LIST: GEOMETRICAL CELLS TO BE USED FOR ASIC DESIGN
CELL LIST

Fig. 9.

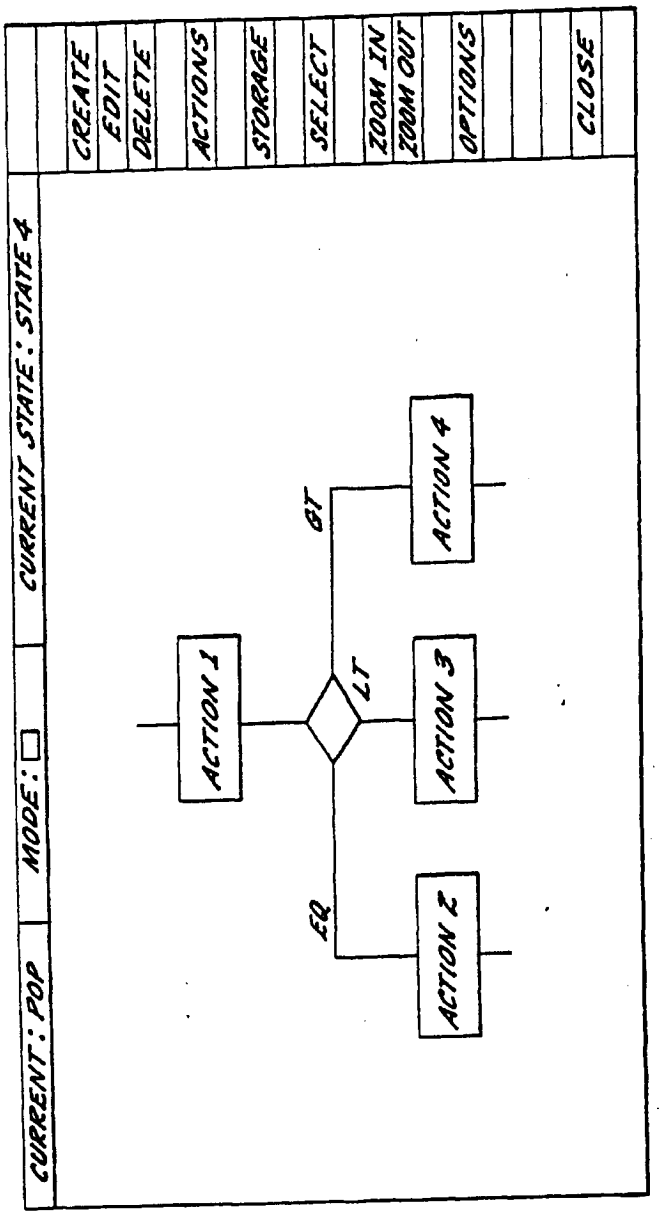

CREATE
EDIT
DELETE
ACTIONS
STORAGE
SELECT
ZOOM IN
ZOOM OUT
OPTIONS
CLOSE
CURRENT STATE : STATE 4
MODE : □
CURRENT : POP
ACTION 1
GT
LT
EQ
ACTION 4
ACTION 3
ACTION 2

FIG. 7.

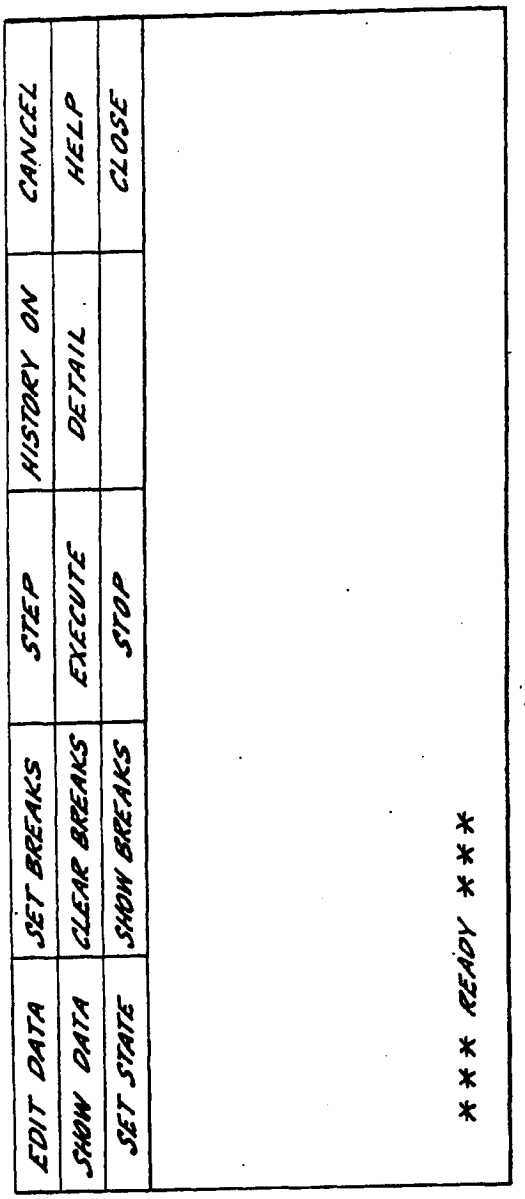

EDIT DATA
SHOW DATA
SET STATE
SET BREAKS
CLEAR BREAKS
SHOW BREAKS
STEP
EXECUTE
STOP
HISTORY ON
DETAIL
CANCEL
HELP
CLOSE
*** READY ***

FIG. 8.

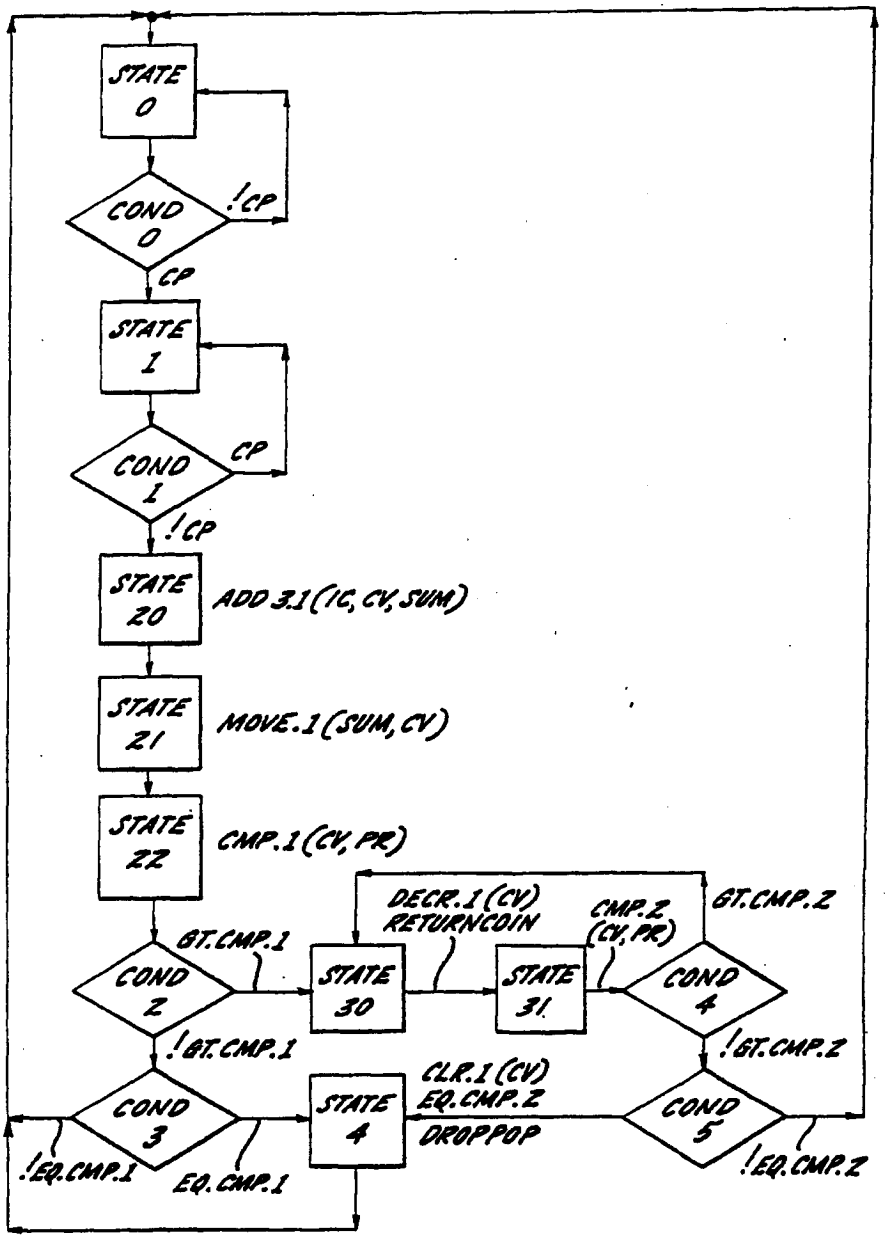

STATE 0
COND 0
!CP
CP
STATE 1
COND 1
CP
!CP
STATE 20
ADD 3.1 (IC, CV, SUM)
STATE 21
MOVE.1 (SUM, CV)
STATE 22
CMP.1 (CV, PR)
GT.CMP.1
COND 2
!GT.CMP.1
STATE 30
DECR.1 (CV)
RETURNCOIN
STATE 31
CMP.2 (CV, PR)
GT.CMP.2
COND 4
!GT.CMP.2
COND 3
!EQ.CMP.1
EQ.CMP.1
STATE 4
CLR.1 (CV)
EQ.CMP.2
DROPPOP
COND 5
!EQ.CMP.2

FIG. 10.

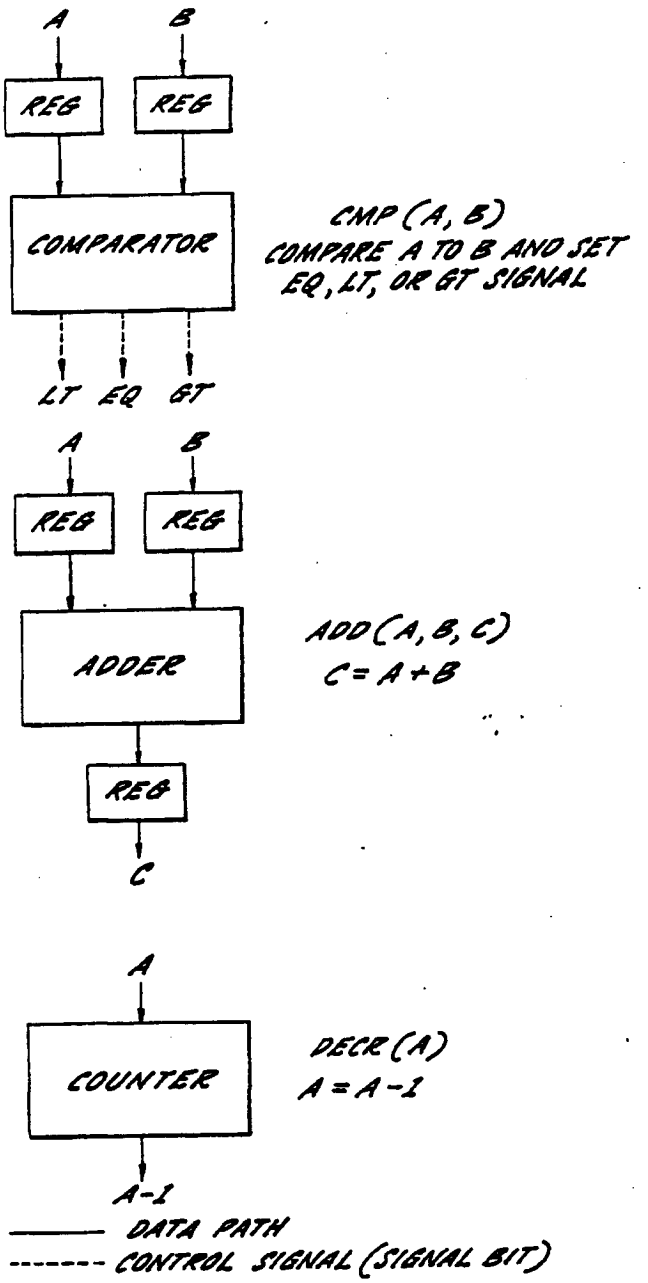

A
B
REG
REG
COMPARATOR
CMP (A, B)
COMPARE A TO B AND SET
EQ, LT, OR GT SIGNAL
LT
EQ
GT
A
B
REG
REG
ADDER
ADD (A, B, C)
C = A + B
REG
C
A
COUNTER
DECR (A)
A = A - 1
A-1
DATA PATH
CONTROL SIGNAL (SIGNAL BIT)

Fig. 11.

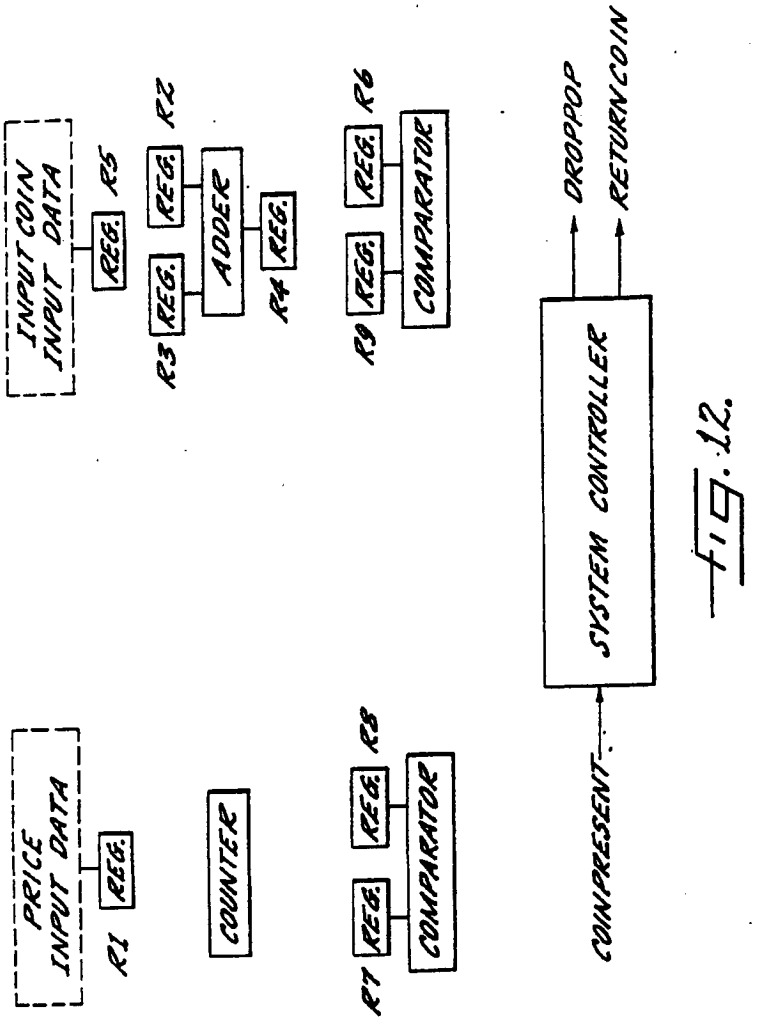

INPUT COIN
INPUT DATA
REG.
R5
REG.
REG.
R2
R3
ADDER
R4
REG.
REG.
R6
REG.
R9
COMPARATOR
DROP POP
RETURN COIN
SYSTEM CONTROLLER
PRICE
INPUT DATA
R1
REG.
COUNTER
REG.
REG.
R8
R7
COMPARATOR
COIN PRESENT

FIG. 12.

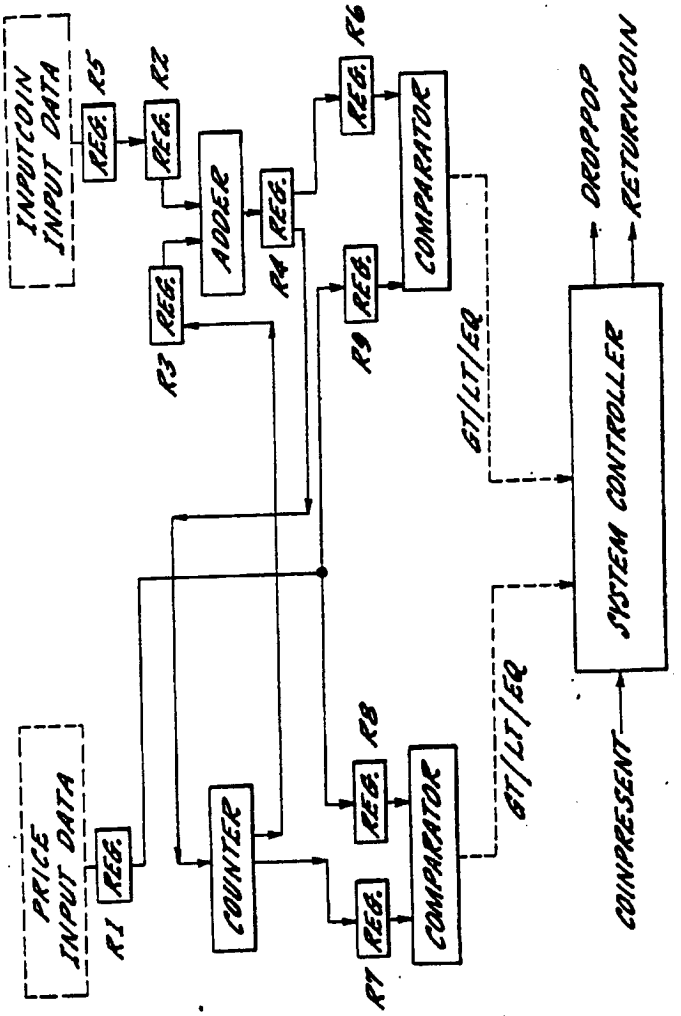

INPUT COIN
INPUT DATA
REG.
R5
R2
ADDER
R4
R6
COMPARATOR
R3
R9
GT/LT/EQ
DROP POP
RETURN COIN
SYSTEM CONTROLLER
PRICE
INPUT DATA
R1
COUNTER
R8
R7
COMPARATOR
GT/LT/EQ
COIN PRESENT

FIG. 13.

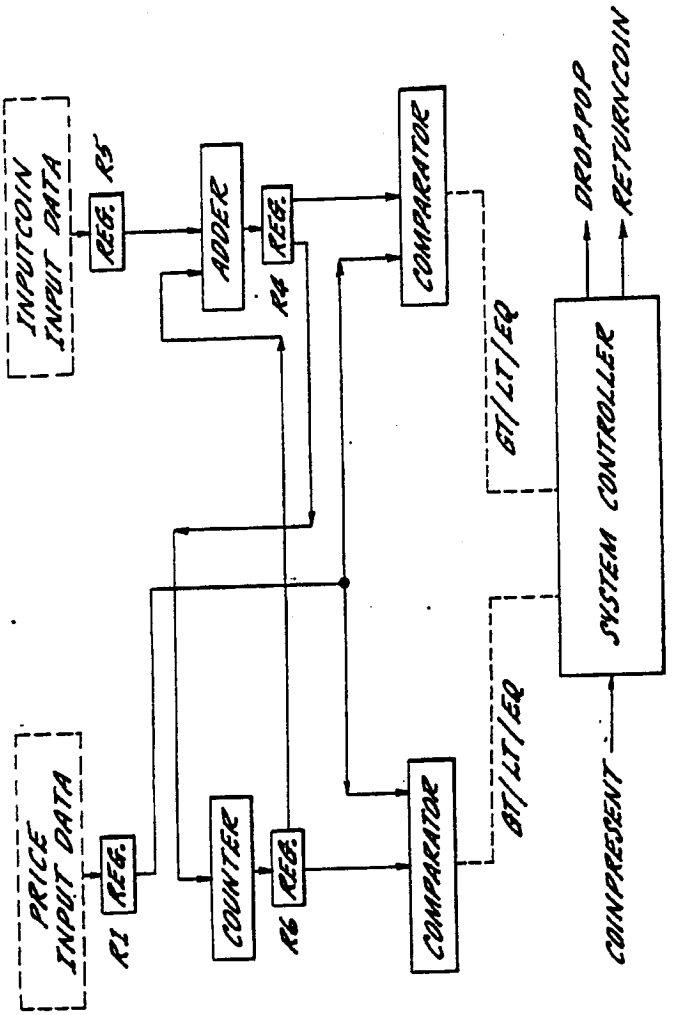

INPUT COIN/ INPUT DATA
REG.
R5
ADDER
REG.
R4
COMPARATOR
GT/LT/EQ
DROP POP
RETURN COIN
SYSTEM CONTROLLER
PRICE INPUT DATA
R1
REG.
COUNTER
R6
REG.
COMPARATOR
GT/LT/EQ
COIN PRESENT

FIG. 14.

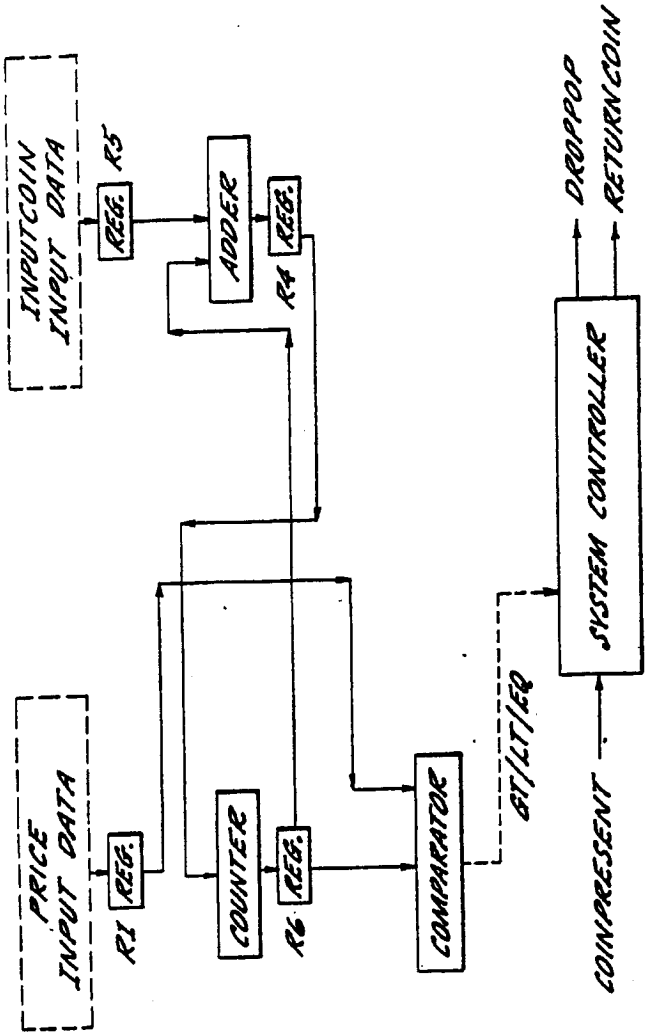

INPUTCOIN
INPUT DATA
REG.
R5
ADDER
REG.
R4
PRICE
INPUT DATA
REG.
R1
COUNTER
REG.
R6
COMPARATOR
GT/LT/EQ
SYSTEM CONTROLLER
DROPPOP
RETURNCOIN
COINPRESENT

FIG. 15.

KNOWLEDGE BASED METHOD AND APPARATUS FOR DESIGNING INTEGRATED CIRCUITS USING FUNCTIONAL SPECIFICATIONS

FIELD AND BACKGROUND OF THE INVENTION

This invention relates to the design of integrated circuits, and more particularly relates to a computer-aided method and apparatus for designing integrated circuits.

An application specific integrated circuit (ASIC) is an integrated circuit chip designed to perform a specific function, as distinguished from standard, general purpose integrated circuit chips, such as microprocessors, memory chips, etc. A highly skilled design engineer having specialized knowledge in VLSI circuit design is ordinarily required to design a ASIC. In the design process, the VLSI design engineer will consider the particular objectives to be accomplished and tasks to be performed by the integrated circuit and will create structural level design specifications which define the various hardware components required to perform the desired function, as well as the interconnection requirements between these components. A system controller must also be designed for synchronizing the operations of these components. This requires an extensive and all encompassing knowledge of the various hardware components required to achieve the desired objectives, as well as their interconnection requirements, signal level compatibility, timing compatibility, physical layout, etc. At each design step, the designer must do tedious analysis. The design specifications created by the VLSI design engineer may, for example, be in the form of circuit schematics, parameters or specialized hardware description languages (HDLs).

From the structural level design specifications, the description of the hardware components and interconnections is converted to a physical chip layout level description which describes the actual topological characteristics of the integrated circuit chip. This physical chip layout level description provides the mask data needed for fabricating the chip.

Due to the tremendous advances in very large scale integration (VLSI) technology, highly complex circuit systems are being built on a single chip. With their complexity and the demand to design custom chips at a faster rate, in large quantities, and for an ever increasing number of specific applications, computer-aided design (CAD) techniques need to be used. CAD techniques have been used with success in design and verification of integrated circuits, at both the structural level and at the physical layout level. For example, CAD systems have been developed for assisting in converting VLSI structural level descriptions of integrated circuits into the physical layout level topological mask data required for actually producing the chip. Although the presently available computer-aided design systems greatly facilitate the design process, the current practice still requires highly skilled VLSI design engineers to create the necessary structural level hardware descriptions.

There is only a small number of VLSI designers who possess the highly specialized skills needed to create structural level integrated circuit hardware descriptions. Even with the assistance of available VLSI CAD tools, the design process is time consuming and the probability of error is also high because of human involvements. There is a very significant need for a better and more cost effective way to design custom integrated circuits.

SUMMARY OF THE INVENTION

In accordance with the present invention a CAD (computer-aided design) system and method is provided which enables a user to define the functional requirements for a desired target integrated circuit, using an easily understood functional architecture independent level representation, and which generates therefrom the detailed information needed for directly producing an application specific integrated circuit (ASIC) to carry out those specific functions. Thus, the present invention, for the first time, opens the possibility for the design and production of ASICs by designers, engineers and technicians who may not possess the specialized expert knowledge of a highly skilled VLSI design engineer.

The functional architecture independent specifications of the desired ASIC can be defined in a suitable manner, such as in list form or preferably in a flowchart format. The flowchart is a highly effective means of describing a sequence of logical operations, and is well understood by software and hardware designers of varying levels of expertise and training. From the flowchart (or other functional specifications), the system and method of the present invention translates the functional architecture independent specifications into structural an architecture specific level definition of an integrated circuit, which can be used directly to produce the ASIC. The structural level definition includes a list of the integrated circuit hardware cells needed to achieve the functional specifications. These cells are selected from a cell library of previously designed hardware cells of various functions and technical specifications. The system also generates data paths among the selected hardware cells. In addition, the present invention generates a system controller and control paths for the selected integrated circuit hardware cells. The list of hardware cells and their interconnection requirements may be represented in the form of a netlist. From the netlist it is possible using either known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level geometrical information (e.g. mask data) required to produce the particular application specific integrated circuit in chip form.

The preferred embodiment of the system and method of the present invention which is described more fully hereinafter is referred to as a Knowledge Based Silicon Compiler (KBSC). The KBSC is an ASIC design methodology based upon artificial intelligence and expert systems technology. The user interface of KBSC is a flowchart editor which allows the designer to represent VLSI systems in the form of a flowchart. The KBSC utilizes a knowledge based expert system, with a knowledge base extracted from expert ASIC designers with a high level of expertise in VLSI design to generate from the flowchart a netlist which describes the selected hardware cells and their interconnection requirements.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be better understood by reference to the detailed description which follows, taken in connection with the accompanying drawings, in which

FIG. 1*a* illustrates a functional level design representation of a portion of a desired target circuit, shown in the form of a flowchart;

FIG. 1*b* illustrates a structural level design representation of an integrated circuit;

FIG. 1*c* illustrates a design representation of a circuit at a physical layout level, such as would be utilized in the fabrication of an integrated circuit chip;

FIG. 2 is a block schematic diagram showing how integrated circuit mask data is created from flowchart descriptions by the KBSC system of the present invention;

FIG. 3 is a somewhat more detailed schematic illustration showing the primary components of the KBSC system;

FIG. 4 is a schematic illustration showing how the ASIC design system of the present invention draws upon selected predefined integrated circuit hardware cells from a cell library;

FIG. 5 is an example flowchart defining a sequence of functional operations to be performed by an integrated circuit;

FIG. 6 is a structural representation showing the hardware blocks and interconnection requirements for the integrated circuit defined in FIG. 5;

FIG. 7 is an illustration of the flowchart editor window;

FIG. 8 is an illustration of the flowchart simulator window;

FIG. 9 is an illustration of the steps involved in cell list generation;

FIG. 10 is an example flowchart for a vending machine system;

FIG. 11 illustrates the hardware components which correspond to each of the three macros used in the flowchart of FIG. 10;

FIG. 12 is an initial block diagram showing the hardware components for an integrated circuit as defined in the flowchart of FIG. 10;

FIG. 13 is a block diagram corresponding to FIG. 12 showing the interconnections between blocks;

FIG. 14 is a block diagram corresponding to FIG. 13 after register optimization; and

FIG. 15 is a block diagram corresponding to FIG. 14 after further optimization.

DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

FIGS. 1*a*, 1*b* and 1*c* illustrate three different levels of representing the design of an integrated circuit. FIG. 1*a* shows a functional (or behavioral) representation architecture independent in the form of a flowchart. A flowchart is a graphic representation of an algorithm and consists of two kinds of blocks or states, namely actions and conditions (decisions). Actions are conventionally represented in the flowchart by a rectangle or box, and conditions are represented by a diamond. Transitions between actions and conditions are represented by lines with arrows. FIG. 1*b* illustrates a structural (or logic) level representation of an integrated circuit. In this representation, blocks are used to represent integrated architecture specific circuit hardware components for performing various functions, and the lines interconnecting the blocks represent paths for the flow of data or control signals between the blocks. The blocks may, for example, represent hardware components such as adders, comparators, registers, system controllers, etc. FIG. 1*c* illustrates a physical layout level representation of an integrated circuit design, which provides the detailed mask data necessary to actually manufacture the devices and conductors which together comprise integrated circuit.

As noted earlier, the design of an integrated circuit at the structural level requires a design engineer with highly specialized skills and expertise in VLSI design. In the KBSC system of the present invention, however, integrated circuits can be designed at a functional level because the expertise in VLSI design is provided and applied by the invention. Allowing the designer to work with flowcharts instead of logic circuit schematics simplifies the task of designing custom integrated circuits, making it quicker, less expensive and more reliable. The designer deals with an algorithm using simple flowcharts at an architecture independent functional (behavioral) level, and needs to know only the necessary logical steps to complete a task, rather than the specific means for accomplishing the task. Designing with flowcharts requires less work in testing because flowcharts allow the designer to work much closer to the algorithm. On the other hand, previously existing VLSI design tools require the designer to represent an algorithm with complex circuit schematics at a structural level, therefore requiring more work in testing. Circuit schematics make it harder for the designer to cope with the algorithm function which needs to be incorporated into the target design because they intermix the hardware and functional considerations. Using flowcharts to design custom integrated circuits will allow a large number of system designers to access VLSI technology, where previously only a small number of designers had the knowledge and skills to create the necessary structural level hardware descriptions.

The overall system flow is illustrated in FIG. 2. The user enters the functional specifications of the circuit into the knowledge based silicon compiler (KBSC) 10 in the form of a flowchart 11. The KBSC 10 then generates a netlist 15 from the flowchart. The netlist 15 includes a custom generated system controller, all other hardware cells required to implement the necessary operations, and interconnection information for connecting the hardware cells and the system controller. The netlist can be used as input to any existing VLSI layout and routing tool 16 to create mask data 18 for geometrical layout.

System Overview

The primary elements or modules which comprise the KBSC system are shown in FIG. 3. In the embodiment illustrated and described herein, these elements or modules are in the form of software programs, although persons skilled in the appropriate art will recognize that these elements can easily be embodied in other forms, such as in hardware.

Referring more particularly to FIG. 3, it will be seen that the KBSC system 10 includes a program 20 called EDSIM, which comprises a flowchart editor 21 for creating and editing flowcharts and a flowchart simulator 22 for simulation and verification of flowcharts. Actions to be performed by each of the rectangles represented in the flowchart are selected from a macro library 23. A program 30 called PSCS (path synthesizer and cell selector) includes a data and control path synthesizer module 31, which is a knowledge based system for data and control path synthesis. PSCS also includes a cell selector 32 for selecting the cells required for system design. The cell selector 32 selects from a cell

library 34 of previously designed hardware cells the appropriate cell or cells required to perform each action and condition represented in the flowchart. A controller generator 33 generates a custom designed system controller for controlling the operations of the other hardware cells. The knowledge base 35 contains ASIC design expert knowledge required for data path synthesis and cell selection. Thus, with a functional flowchart input, PSCS generates a system controller, selects all other hardware cells, generates data and control paths, and generates a netlist describing all of this design information.

The KBSC system employs a hierarchal cell selection ASIC design approach, as is illustrated in FIG. 4. Rather than generating every required hardware cell from scratch, the system draws upon a cell library 34 of previously designed, tested and proven hardware cells of various types and of various functional capabilities with a given type. The macro library 23 contains a set of macros defining various actions which can be specified in the flowchart. For each macro function in the macro library 23 there may be several hardware cells in the cell library 34 of differing geometry and characteristics capable of performing the specified function. Using a rule based expert system with a knowledge base 35 extracted from expert ASIC designers, the KBSC system selects from the cell library 34 the optimum cell for carrying out the desired function.

Referring again to FIG. 3, the cells selected by the cell selector 32, the controller information generated by the controller generator 33 and the data and control paths generated by the data/control path synthesizer 31 are all utilized by the PSCS program 30 to generate the netlist 15. The netlist is a list which identifies each block in the circuit and the interconnections between the respective inputs and outputs of each block. The netlist provides all the necessary information required to produce the integrated circuit. Computer-aided design systems for cell placement and routing are commercially available which will receive netlist data as input and will lay out the respective cells in the chip, generate the necessary routing, and produce mask data which can be directly used by a chip foundry in the fabrication of integrated circuits.

System Requirements

The KBSC system can be operated on a suitable programed general purpose digital computer. By way of example, one embodiment of the system is operated in a work station environment such as Sun3 and VAXStation-II/GPX Running UNIX Operating System and X Window Manager. The work station requires a minimum of 8 megabytes of main storage and 20 megabytes of hard disk space. The monitor used is a color screen with 8-bit planes. The software uses C programming language and INGRES relational data base.

The human interface is mainly done by the use of pop up menus, buttons, and a special purpose command language. The permanent data of the integrated circuit design are stored in the data base for easy retrieval and update. Main memory stores the next data temporarily, executable code, design data (flowchart, logic, etc.), data base (cell library), and knowledge base. The CPU performs the main tasks of creating and simulating flowcharts and the automatic synthesis of the design.

Flowchart Example

To describe the mapping of a flowchart to a netlist, consider an example flowchart shown in FIG. 5, which is of part of a larger overall system. In this illustrative flowchart, two variables, VAL1 and VAL2 are compared and if they are equal, they are added together. In this instance, the first action (Action 1) involves moving the value of variable VALI to register A. The second action comprises moving the value of variable VAL2 to register B. Condition 1 comprises comparing the values in registers A and B. Action 3 comprises adding the values of registers A and B and storing the result in register C.

In producing an integrated circuit to carry out the function defined in FIG. 5, the KBSC maps the flowchart description of the behavior of the system to interconnection requirements between hardware cells. The hardware cells are controlled by a system controller which generates all control signals. There are two types of variables involved in a system controller:

(1) Input variables: These are generated by hardware cells, and/or are external input to the controller. These correspond to conditions in the flowchart.

(2) Output variables: These are generated by the system controller and correspond to actions in the flowchart.

FIG. 6 illustrates the results of mapping the flowchart of FIG. 5 onto hardware cells. The actions and the conditions in the flowchart are used for cell selection and data and control path synthesis. The VALI register and VAL2 register and the data paths leading therefrom have already been allocated in actions occurring before Action 1 in our example. Action 1 causes generation of the data register A. Similarly, Action 2 causes the allocation of data register B. The comparator is allocated as a result of the comparison operation in Condition 1. The comparison operation is accomplished by (1) selecting a comparator cell, (2) mapping the inputs of the comparator cell to registers A and B, (3) generating data paths to connect the comparator with the registers A and B and (4) generating input variables corresponding to equal to, greater than, and less than for the system controller. Similarly the add operation in Action 3 causes selection of the adder cell, mapping of the adder parameters to the registers and creating the data paths.

Following this methodology, a block list can be generated for a given flowchart. This block list consists of a system controller and as many other blocks as may be required for performing the necessary operations. The blocks are connected with data paths, and the blocks are controlled by the system controller through control paths. These blocks can be mapped to the cells selected from a cell library to produce a cell list.

Interactive Flowchart Editor and Simulator

The creation and verification of the flowchart is the first step in the VLSI design methodology. The translation from an algorithm to an equivalent flowchart is performed with the Flowchart Editor 21 (FIG. 3). The verification of the edited flowchart is performed by the Flowchart Simulator 22 The Flowchart Editor and Simulator are integrated into one working environment for interactive flowchart editing, with a designer friendly interface.

EDSIM is the program which contains the Flowchart Editor 21 and the Flowchart Simulator 22. It also provides functions such as loading and saving flow-

charts. EDSIM will generate an intermediate file, called a statelist, for each flowchart. This file is then used by the PSCS program 30 to generate a netlist.

Flowchart Editor

The Flowchart Editor 21 is a software module used for displaying, creating, and editing the flowchart. This module is controlled through the flowchart editing window illustrated in FIG. 7. Along with editing functions the Flowchart Editor also provides checking of design errors.

The following is a description of the operations of the Flowchart Editor. The main editing functions include, create, edit, and delete states, conditions, and transitions. The create operation allows the designer to add a new state, condition, or transitions to a flowchart. Edit allows the designer to change the position of a state, condition or transition, and delete allows the designer to remove a state, condition or transition from the current flowchart. States which contain actions are represented by boxes, conditions are represented by diamonds, and transitions are represented by lines with arrows showing the direction of the transition.

Edit actions allows the designer to assign actions to each box. These actions are made up of macro names and arguments. An example of arguments is the setting and clearing of external signals. A list of basic macros available in the macro library 23 is shown in Table 1.

TABLE 1

Macro	Description
ADD (A,B,C)	C = A + B
SUB (A,B,C)	C = A − B
MULT (A,B,C)	C = A * B
DIV (A,B,C)	C = A div B
DECR (A)	A = A − 1
INCR (A)	A = A + 1
CLR (A)	A = 0
REG (A,B)	B = A
CMP (A,B)	Compare A to B and set EQ,LT,GT signals
CMP0 (A)	Compare A to 0 and set EQ,LT,GT signals
NEGATE (A)	A = NOT (A)
MOD (A,B,C)	C = A Modulus B
POW (A,B,C)	C = A B
DC2 (A,S1,S2,S3,S4)	Decode A into S1,S2,S3,S4
EC2 (S1,S2,S3,S4,A)	Encode S1,S2,S3,S4 into A
MOVE (A,B)	B = A
CALL sub-flowchart (A,B, . . .)	Call a sub-flowchart. Pass A,B . . .
START (A,B, . . .)	Beginning state of a sub-flowchart
STOP (A,B . . .)	Ending state of a sub-flowchart

The Flowchart Editor also provides a graphical display of the flowchart as the Flowchart Simulator simulates the flowchart. This graphical display consists of boxes, diamonds, and lines as shown in FIG. 7. All are drawn on the screen and look like a traditional flowchart. By displaying the flowchart on the screen during simulation it allows the designer to design and verify the flowchart at the same time.

Flowchart Simulator

The Flowchart Simulator 22 is a software module used for simulating flowcharts. This module is controlled through the simulator window illustrated in FIG. 8. The Flowchart Simulator simulates the transitions between states and conditions in a flowchart. The following is a list of the operations of the Flowchart Simulator:

edit data—Change the value of a register or memory.
set state—Set the next state to be simulated.
set detail or summary display—Display summary or detail information during simulation.
set breaks—Set a breakpoint.
clear breaks—Clear all breakpoints.
show breaks—Display current breakpoints.
step—Step through one transition.
execute—Execute the flowchart.
stop—Stop executing of the flowchart. history ON or history OFF—Set history recording on or off.
cancel—Cancel current operation.
help—Display help screen.
close—Close the simulator window.

The results of the simulation are displayed within the simulator window. Also the editor window will track the flowchart as it is being simulated. This tracking of the flowchart makes it easy to edit the flowchart when an error is found.

Cell Selection

The Cell Selector 32 is a knowledge based system for selecting a set of optimum cells from the cell library 34 to implement a VLSI system. The selection is based on functional descriptions in the flowchart, as specified by the macros assigned to each action represented in the flowchart. The cells selected for implementing a VLSI system depend on factors such as cell function, fabrication technology used, power limitations, time delays etc. The cell selector uses a knowledge base extracted from VLSI design experts to make the cell selection.

To design a VLSI system from a flowchart description of a user application, it is necessary to match the functions in a flowchart with cells from a cell library. This mapping needs the use of artificial intelligence techniques because the cell selection process is complicated and is done on the basis of a number of design parameters and constraints. The concept used for cell selection is analogous to that used in software compilation. In software compilation a number of subroutines are linked from libraries. In the design of VLSI systems, a functional macro can be mapped to library cell.

FIG. 4 illustrates the concept of hierarchical cell selection. The cell selection process is performed in two steps:

(1) selection of functional macros
(2) selection of geometrical cells

A set of basic macros is shown in Table 1. A macro corresponds to an action in the flowchart. As an example, consider the operation of adding A and B and storing the result in C. This function is mapped to the addition macro ADD(X, Y, Z). The flowchart editor and flowchart simulator are used to draw the rectangles, diamonds and lines of the flowchart, to assign a macro selected from the macro library 23 to each action represented in the flowchart, and to verify the functions in flowcharts. The flowchart is converted into an intermediate form (statelist) and input to the Cell Selector.

The Cell Selector uses a rule based expert system to select the appropriate cell or cells to perform each action. If the cell library has a number of cells with different geometries for performing the operation specified by the macro, then an appropriate cell can be selected on the basis of factors such as cell function, process technology used, time delay, power consumption, etc.

The knowledge base of Cell Selector 32 contains information (rules) relating to:

(1) selection of macros
(2) merging two macros

(3) mapping of macros to cells
(4) merging two cells
(5) error diagnostics

The above information is stored in the knowledge base 35 as rules.

Cell List Generation

FIG. 9 shows the cell list generation steps. The first step of cell list generation is the transformation of the flowchart description into a structure that can be used by the Cell Selector. This structure is called the statelist. The blocklist is generated from the statelist by the inference engine. The blocklist contains a list of the functional blocks to be used in the integrated circuit. Rules of the following type are applied during this stage.

map arguments to data paths
map actions to macros
connect these blocks

Rules also provide for optimization and error diagnostics at this level.

The cell selector maps the blocks to cells selected from the cell library 34. It selects an optimum cell for a block. This involves the formulation of rules for selecting appropriate cells from the cell library. Four types of information are stored for each cell. These are:

(1) functional level information: description of the cell at the register transfer level.
(2) logic level information: description in terms of flip-flops and gates.
(3) circuit level information: description at the transistor level.
(4) Layout level information: geometrical mask level specification.

The attributes of a cell are:

cell name
description
function
width
height
status
technology
minimum delay
typical delay
maximum delay
power
file
designer
date
comment
inspector

In the cell selection process, the above information can be used. Some parameters that can be used to map macros to cells are:

(1) name of macro
(2) function to be performed
(3) complexity of the chip
(4) fabrication technology
(5) delay time allowed
(6) power consumption
(7) bit size of macro data paths

Netlist Generation

The netlist is generated after the cells have been selected by PSCS. PSCS also uses the macro definitions for connecting the cell terminals to other cells. PSCS uses the state-to-state transition information from an intermediate form representation of a flowchart (i.e. the statelist) to generate a netlist. PSCS contains the following knowledge for netlist generation:

(1) Data path synthesis
(2) Data path optimization
(3) Macro definitions
(4) Cell library
(5) Error detection and correction

The above information is stored in the knowledge base 35 as rules. Knowledge engineers help in the formulation of these rules from ASIC design experts. The macro library 23 and the cell library 34 are stored in a database of KBSC.

A number of operations are performed by PSCS. The following is a top level description of PSCS operations:

(1) Read the flowchart intermediate file and build a statelist.
(2) current_context=START
(3) Start the inference engine and load the current context rules.
(4) Perform one of the following operations depending upon current_context:
  (a) Modify the statelist for correct implementation.
  (b) Create blocklist, macrolist and data paths.
  (c) Optimize blocklist and datapath list and perform error checks.
  (d) Convert blocks to cells.
  (e) Optimize cell list and perform error checks.
  (f) Generate netlist.
  (g) Optimize netlist and perform error checks and upon completion Goto 7.
(5) If current_context has changed, load new context rules.
(6) Goto 4.
(7) Output netlist file and stf files and Stop.

In the following sections, operations mentioned in step 4 are described. The Rule Language and PSCS display are also described.

Rule Language

The rule language of PSCS is designed to be declarative and to facilitate rule editing. In order to make the expert understand the structure of the knowledge base, the rule language provides means for knowledge representation. This will enable the format of data structures to be stated in the rule base, which will enable the expert to refer to them and understand the various structures used by the system. For example, the expert can analyze the structure of wire and determine its components. The expert can then refer these components into rules. If a new object has to be defined, then the expert can declare a new structure and modify some existing structure to link to this new structure. In this way, the growth of the data structures can be visualized better by the expert. This in turn helps the designer to update and append rules.

The following features are included in the rule language:

(i) Knowledge representation in the form of a record structure.
(ii) Conditional expressions in the antecedent of a rule.
(iii) Facility to create and destroy structure in rule actions.
(iv) The assignment statement in the action of a rule.
(v) Facility for input and output in rule actions.
(vi) Provide facility to invoke C functions from rule actions.

The rule format to be used is as follows:

The rule format to be used is as follows:

```
Rule    <number> <context>
If {
        <if-clause>
}
Then    {
        <then-clause>
}
where   <number>        rule number
        <context>       context in which this rule is active
        <if-clause>     the condition part of the rule
        <then-clause>   the action part of the rule
```

Inference Strategy

The inference strategy is based on a fast pattern matching algorithm. The rules are stored in a network and the requirement to iterate through the rules is avoided. This speeds up the execution. The conflict resolution strategy to be used is based on the following:

(1) The rule containing the most recent data is selected.

(2) The rule which has the most complex condition is selected.

(3) The rule declared first is selected.

Rule Editor

PSCS provides an interactive rule editor which enables the expert to update the rule set. The rules are stored in a database so that editing capabilities of the database package can be used for rule editing. To perform this operation the expert needs to be familiar with the various knowledge structures and the inferencing process. If this is not possible, then the help of a knowledge engineer is needed.

PSCS provides a menu from which various options can be set. Mechanisms are provided for setting various debugging flags and display options, and for the overall control of PSCS.

Facility is provided to save and display the blocklist created by the user. The blocklist configuration created by the user can be saved in a file and later be printed with a plotter. Also the PSCS display can be reset to restart the display process.

PSCS Example Rules:

Rule 1		
	IF	no blocks exist
	THEN	generate a system controller.
Rule 2		
	IF	a state exists which has a macro AND this macro has not been mapped to a block
	THEN	find a corresponding macro in the library and generate a block for this macro.
Rule 3		
	IF	there is a transition between two states AND there are macros in these states using the same argument
	THEN	make a connection from a register corresponding to the first macro to another register corresponding to the second macro.
Rule 4		
	IF	a register has only a single connection from another register
	THEN	combine these registers into a single register.
Rule 5		
	IF	there are two comparators AND input data widths are of the same size AND one input of these is same AND the outputs of the comparators are used to perform the same operation.
	THEN	combine these comparators into a single comparator.
Rule 6		
	IF	there is a data without a register
	THEN	allocate a register for this data.
Rule 7		
	IF	all the blocks have been interconnected AND a block has a few terminals not connected
	THEN	remove the block and its terminals, or issue an error message.
Rule 8		
	IF	memory is to be used, but a block has not been created for it
	THEN	create a memory block with data, address, read and write data and control terminals.
Rule 9		
	IF	a register has a single connection to a counter
	THEN	combine the register and the counter; remove the register and its terminals.
Rule 10		
	IF	there are connections to a terminal of a block from many different blocks
	THEN	insert a multiplexor; remove the connections to the terminals and connect them to the input of the multiplexor; connect the output of the multiplexor to the input of the block.

Additional rules address the following points:
remove cell(s) that can be replaced by using the outputs of other cell(s)
reduce multiplexor trees
use fan-out from the cells, etc.

Soft Drink Vending Machine Controller Design Example

The following example illustrates how the previously described features of the present invention are employed in the design of an application specific integrated circuit (ASIC). In this illustrative example the ASIC is designed for use as a vending machine controller. The vending machine controller receives a signal each time a coin has been deposited in a coin receiver. The coin value is recorded and when coins totalling the correct amount are received, the controller generates a signal to dispense a soft drink. When coins totalling more than the cost of the soft drink are received, the controller dispenses change in the correct amount.

This vending machine controller example is patterned after a textbook example used in teaching digital system controller design. See Fletcher, William I., *An Engineering Approach to Digital Design*, Prentice-Hall, Inc., pp. 491–505. Reference may be made to this textbook example for a more complete explanation of this vending machine controller requirements, and for an understanding and appreciation of the complex design procedures prior to the present invention for designing the hardware components for a controller.

FIG. 10 illustrates a flowchart for the vending machine controller system. This flowchart would be entered into the KBSC system by the user through the flowchart editor. Briefly reviewing the flowchart, the controller receives a coin present signal when a coin is received in the coin receiver. State0 and cond0 define a waiting state awaiting deposit of a coin. The symbol CP represents "coin present" and the symbol !CP repre-

sents "coin not present". Statel and condI determine when the coin has cleared the coin receiver. At state20, after receipt of a coin, the macro instruction ADD3.1 (lc, cv, sum) instructs the system to add lc (last coin) and cv (coin value) and store the result as sum. The macro instruction associated with state21 moves the value in the register sum to cv. The macro CMP.1 at state22 compares the value of cv with PR (price of soft drink) and returns signals EQ, GT and LT. The condition cond2 tests the result of the compare operation CMP.1. If the result is "not greater than" (!GT.CMP.1), then the condition cond3 tests to see whether the result is "equal" (EQ.CMP.1). If the result is "not equal" (!EQ.CMP.1), then control is returned to state0 awaiting the deposit of another coin. If cond3 is EQ, then state4 generates a control signal to dispense a soft drink (droppop) and the macro instruction CLR.1(cv) resets cv to zero awaiting another customer.

If the total coins deposited exceed the price, then state30 produces the action "returncoin". Additionally, the macro DECR.1 (cv) reduces the value of cv by the amount of the returned coin. At state31 cv and PR are again compared. If cv is still greater than PR, then control passes to state30 for return of another coin. The condition cond5 tests whether the result of CMP.2 is EQ and will result in either dispensing a drink (droppop) true or branching to state0 awaiting deposit of another coin. The macros associated with the states shown in FIG. 10 correspond to those defined in Table 1 above and define the particular actions which are to be performed at the respective states.

Appendix A shows the intermediate file or "statelist" produced from the flowchart of FIG. 10. This statelist is produced as output from the EDSIM program 20 and is used as input to the PSCS program 30 (FIG. 3).

FIG. 11 illustrates for each of the macros used in the flowchart of FIG. 10, the corresponding hardware blocks. It will be seen that the comparison macro CMP (A,B) results in the generation of a register for storing value A, a register for storing value B, and a comparator block and also produces control paths to the system controller for the EQ, LT, and GT signals generated as a result of the comparison operation. The addition macro ADD (A,B,C) results in the generation of a register for each of the input values A and B, a register for the output value C, and in the generation of an adder block. The macro DECR (A) results in the generation of a counter block. The PSCS program 30 maps each of the macros used in the flowchart of FIG. 10 to the corresponding hardware components results in the generation of the hardware blocks shown in FIG. 12. In generating the illustrated blocks, the PSCS program 30 relied upon rules 1 and 2 of the above listed example rules.

FIG. 13 illustrates the interconnection of the block of FIG. 12 with data paths and control paths. Rule 3 was used by the data/control path synthesizer program 31 in mapping the data and control paths.

FIG. 14 shows the result of optimizing the circuit by applying rule 4 to eliminate redundant registers. As a result of application of this rule, the registers R2, R3, R7, R8, and R9 in FIG. 13 were removed. FIG. 15 shows the block diagram after further optimization in which redundant comparators are consolidated. This optimization is achieved in the PSCS program 30 by application of rule 5.

Having now defined the system controller block, the other necessary hardware blocks and the data and control paths for the integrated circuit, the PSCS program 30 now generates a netlist 15 defining these hardware components and their interconnection requirements. From this netlist the mask data for producing the integrated circuit can be directly produced using available VLSI CAD tools.

```
name rpop;
data path @ic<0:5>, cv<0:5>, sum<0:5>, @pr<0:5>;
{
state4 : state0;
state30 : state31;
state21 : state22;
state20 : state21;
state0 :. lcp state0;
state0 :. cp state1;
state1 :. cp state1;
state1 :. lcp state20;
state22 :. GT.CMP.1 state30;
state22 :. !GT.CMP.1*EQ.CMP.1 state4;
state22 :. !GT.CMP.1*!EQ.CMP.1 state0;
state31 :. GT.CMP.2 state30;
state31 :. !GT.CMP.2*EQ.CMP.2 state4;
state31 :. !GT.CMP.2*!EQ.CMP.2 state0;
state30 :: returncoin;
state30 :: DECR.1(cv);
state4 :: droppop;
state4 :: CLR.1(cv);
state31 :: CMP.2(cv,pr);
state22 :: CMP.1(cv,pr);
state21 :: MOVE.1(sum,cv);
state20 :: ADD3.1(ic,cv,sum);
}
```

That which I claimed is:

1. A computer-aided design system for designing an application specific integrated circuit directly from architecture independent functional specifications for the integrated circuit, comprising

a macro library defining a set of architecture independent operations comprised of actions and conditions;

input specification means operable by a user for defining architecture independent functional specifications for the integrated circuit, said functional specifications being comprised of a series of operations comprised of actions and conditions, said input specification means including means to permit the user to specify for each operation a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library;

cell selection means for selecting from said cell library for each macro specified by said input specification means, appropriate hardware cells for performing the operation defined by the specified macro, said cell selection means comprising an expert system including a knowledge base containing rules for. selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base; and

netlist generator means cooperating with said cell selection means for generating as output from the system a netlist defining the hardware cells which are needed to achieve the functional requirements of the integrated circuit and the connections therebetween.

2. The system as defined in claim 1 wherein said input means comprises means specification for receiving user

input of a list defining the series of actions and conditions.

3. The system as defined in claim 1 additionally including mask data generator means for generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

4. The system as defined in claim 1 wherein said input means comprises flowchart editor means specification for creating a flowchart having elements representing said series of actions and conditions.

5. The system as defined in claim 4 additionally including flowchart simulator means for simulating the functions defined in the flowchart to enable the user to verify the operation of the integrated circuit.

6. The system as defined in claim 1 additionally including data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selection means.

7. The system as defined in claim 6 wherein said data path generator means comprises a knowledge base containing rules for selecting data paths between hardware cells and inference engine means for selecting data paths between the hardware cells selected by said cell selection means in accordance with the rules of said knowledge base and the arguments of the specified macros.

8. The system as defined in claim 6 additionally including control generator means for generating a controller and control paths for the hardware cells selected by said cell selection means.

9. A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining architecture independent functional requirements of the integrated circuit comprising

a marco library defining a set of architecture independent operations comprised of actions and conditions;

flowchart editor means operable by a user for creating a flowchart having elements representing said architecture independent operations;

said flowchart editor means including macro specification means for permitting the user to specify for each operation represented in the flowchart a macro selected from said macro library;

a cell library defining a set of available integrated circuit hardware cells for performing the available operations defined in said macro library;

cell selection means for selecting form said cell library for each specified macro, appropriate hardware cells for performing the operation defined by the specified macro, said cell selection means comprising an expert system including a knowledge base containing rules for selecting hardware cells from said cell library and inference engine means for selecting appropriate hardware cells from said cell library in accordance with the rules of said knowledge base; and

data path generator means cooperating with said cell selection means for generating data paths for the hardware cells selected by said cell selector means, said data path generator means comprising a knowledge base containing rules for selecting data paths between hardware cells and inference engine means for selecting data paths between hardware cells selected by said cell selection means in accordance with the rules of said knowledge base and the arguments of the specified macros.

10. The system as defined in claim 9 additionally including control generator means for generating a controller and control paths for the hardware cells selected by said cell selection means.

11. A computer-aided design system for designing an application specific integrated circuit directly from a flowchart defining architecture independent functional requirements of the integrated circuit, comprising

flowchart editor means operable by a user for creating a flowchart having boxes representing architecture independent actions, diamonds representing architecture independent conditions, and lines with arrows representing transitions between actions and condition and including means for specifying for each box or diamond, a particular action or condition to be performed;

a cell library defining a set of available integrated circuit hardware cells for performing actions and conditions;

a knowledge base containing rules for selecting hardware cells from said cell library and for generating data and control paths for hardware cells; and

expert system means operable with said knowledge base for translating the flowchart defined by said flowchart editor means into a netlist defining the necessary hardware cells and data and control paths required in an integrated circuit having the specified functional requirements.

12. The system as defined in claim 11 including mask data generator means for generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

13. A computer-aided design process for designing an application specific integrated circuit which will perform a desired function comprising

storing a set of definitions of architecture independent actions and conditions;

storing data describing a set of available integrated circuit hardware cells for performing the actions and conditions defined in the stored set;

storing in an expert system knowledge base a set of rules for selecting hardware cells to perform the actions and conditions;

describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions;

specifying for each described action and condition of the series one of said stored definitions which corresponds to the desired action or condition to be performed; and

selecting from said stored data for each of the specified definitions a corresponding integrated circuit hardware cell for performing the desired function of the application specific integrated circuit, said step of selecting a hardware cell comprising applying to the specified definition of the action or condition to be performed, a set of cell selection rules stored in said expert system knowledge base and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.

14. A process as defined in claim 13, including generating from the netlist the mask data required to produce an integrated circuit having the desired function.

15. A process as defined in claim 13 including the further step of generating data paths for the selected integrated circuit hardware cells.

16. A process as defined in claim 15 wherein said step of generating data paths comprises applying to the selected cells a set of data path rules stored in a knowledge base and generating the data paths therefrom.

17. A process as defined in claim 16 including the further step of generating control paths for the selected integrated circuit hardware cells.

18. A knowledge based design process for designing an application specific integrated circuit which will perform a desired function comprising

storing in a macro library a set of macros defining architecture independent actions and conditions;

storing in a cell library a set of available integrated circuit hardware cells for performing the actions and conditions;

storing in a knowledge base set of rules for selecting hardware cells from said cell library to perform the actions and conditions defined by the stored macros;

describing for a proposed application specific integrated circuit a flowchart comprised of elements representing a series of architecture independent actions and conditions which carry out the function to be performed by the integrated circuit;

specifying for each described action and condition of said series a macro selected from the macro library which corresponds to the action or condition; and

applying rules of said knowledge base to the specified macros to select from said cell library the hardware cells required for performing the desired function of the application specific integrated circuit and generating for the selected integrated circuit hardware cells, a netlist defining the hardware cells which are needed to perform the desired function of the integrated circuit and the interconnection requirements therefor.

19. A process as defined in claim 18 also including the steps of

storing in said knowledge base a set of rules for creating data paths between hardware cells, and

applying rules of said knowledge base to the specified means to create data paths for the selected hardware cells.

20. A process as defined in claim 19 also including the steps of generating a controller and generating control paths for the selected hardware cells.

* * * * *

Gary M. Hoffman (*Pro Hac Vice*)
Kenneth W. Brothers (*Pro Hac Vice*)
DICKSTEIN SHAPIRO MORIN & OSHINSKY, LLP
2101 L Street, NW
Washington, DC 20037-1526
Phone (202) 785-9700
Fax (202) 887-0689

Edward A. Meilman (*Pro Hac Vice*)
DICKSTEIN SHAPIRO MORIN & OSHINSKY, LLP
1177 Avenue of the Americas
New York, New York 10036-2714
Phone (212) 835-1400
Fax (212) 997-9880

Jeffrey B. Demain, State Bar No. 126715
Jonathan Weissglass, State Bar No. 185008
ALTSHULER, BERZON, NUSSBAUM, RUBIN & DEMAIN
177 Post Street, Suite 300
San Francisco, California 94108
Phone (415) 421-7151
Fax (415) 362-8064

Attorneys for Ricoh Company, Ltd.

Teresa M. Corbin (SBN 132360)
Christopher Kelley (SBN 166608)
Thomas C. Mavrakakis (SBN 177927)
HOWREY SIMON ARNOLD & WHITE
301 Ravenswood Avenue
Menlo Park, California 94025
Telephone: (650) 463-8100
Facsimile: (650) 463-8400

Attorneys for Synopsys, Inc. and Aeroflex Incorporated, et al.

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

RICOH COMPANY, LTD., Plaintiff, vs. AEROFLEX INCORPORATED, et al., Defendants	**CASE NO. C-03-4669-MJJ (EMC)** **CASE NO. C-03-2289-MJJ (EMC)**
SYNOPSYS, INC., Plaintiff, vs. RICOH COMPANY, LTD., Defendant	**MANUAL FILING NOTIFICATION RE ATTACHMENT 3 TO EXHIBIT A OF JOINT CLAIM CONSTRUCTION AND PREHEARING STATEMENT** **Date: October 20, 2004** **Time: 2:30 p.m.** **Courtroom: 11**

MANUAL FILING NOTIFICATION

Regarding: Attachment 3 to Exhibit A of Joint Claim Construction and Prehearing Statement

This filing is in paper or physical form only, and is being maintained in the case file in the Clerk's office.

If you are a participant in this case, this filing will be served in hard-copy shortly.

For information on retrieving this filing directly from the court, please see the court's main web site at http://www.cand.uscourts.gov under Frequently Asked Questions (FAQ).

This filing was not efiled for the following reason(s):

☒ Voluminous Document (PDF file size larger than the efiling system allows)

___ Unable to Scan Documents

___ Physical Object (description): _____

___ Non Graphical/Textual Computer File (audio, video, etc.) on CD or other media

___ Item Under Seal

___ Conformance with the Judicial Conference Privacy Policy (General Order 53).

___ Other (description): _____

4

United States Patent [19]

Darringer et al.

[11] **Patent Number: 4,703,435**

[45] **Date of Patent: Oct. 27, 1987**

[54] **LOGIC SYNTHESIZER**

[75] Inventors: **John A. Darringer**, Mahopac; **William H. Joyner, Jr.**, Katonah, both of N.Y.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **631,364**

[22] Filed: **Jul. 16, 1984**

[51] **Int. Cl.**[4] **G06F 7/00; G06F 15/60**

[52] **U.S. Cl.** **364/489**; 364/300; 364/488

[58] **Field of Search** 364/488, 489, 490, 491, 364/300; 307/303

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,377,849	3/1983	Finger et al.	364/491
4,580,228	4/1986	Noto	364/300 X
4,591,993	5/1986	Griffin et al.	364/491
4,612,618	9/1986	Pryor et al.	364/488 X

FOREIGN PATENT DOCUMENTS

0168650	1/1986	European Pat. Off. .	

OTHER PUBLICATIONS

Friedman et al., "Methods used in an Automatic Logic Design Generator (Alert)", *IEEE Trans. Comp. C*-18, pp. 593–614, 1969.

Mitchell et al., "The Use of High Speed Proms to Generate Boolean Functions,", *Wescon Technical Papers*, Sep. 1978, pp. 1-7.

Mano, "Digital Logic and Computer Design", Ch. 3, pp. 72–103, 1979.

Introduction to the Automated Synthesis of Computer; Herbert Schorr; Department of Electrical Engineering Digital Systems.

Laboratory Technical Report No. 16 (Mar. 1962). Ph.D. Thesis, Princeton University.

Minimization of Boolean Functions; F. J. Hill et al.; "Introduction to Switch Theory and Logical Design", 1973.

The Description, Simulation, and Automatic Implementation of Digital Computer Processors; John A. Darringer.

The Experimental Compiling System; F. E. Allen; IBM J. Res. Development; vol. 24, No. 6, Nov. 1980.

Logic Synthesis Through Local Transformation; John A. Darringer; IBM Journal of Research and Development; vol. 25, No. 4, Jul. 1981.

Programming Language; Yaohan Chu; vol. 8, No. 10, 10/65.

Development and Application of a Designer Oriented Cyclic Simulator; G. J. Parasch; 13th DA Conference 1976.

Logic Synthesis; Melvin A. Breuer; M. Breuer "Design Automation of Digital Systems", Prentice–Hall 1972.

Synthesis of Combinational Logic Networks; D. L. Dietmeyer "Logic Design of Digital Systems"; Allyn & Bacon, Boston 1978.

Quality of Designs from an Automatic Logic Generator (Alert)*; Theodore D. Friedman and Sih–Chin Yang; 7th DA Conference 1970.

On Logic Comparison; Leonard Berman; 18th DA Conference 1981.

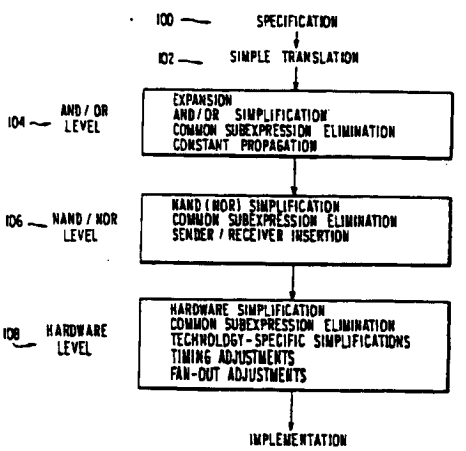
(List continued on next page.)

Primary Examiner—Errol A. Krass
Assistant Examiner—Joseph L. Dixon
Attorney, Agent, or Firm—Sughrue, Mion, Zinn, Macpeak, and Seas

[57] **ABSTRACT**

Logic is synthesized from a flowchart-level description by first generating an AND/OR logic design, simplifying the AND/OR logic, converting the AND/OR logic to NAND or NOR logic, applying particular sequences of simplifying transformations to the NAND or NOR logic, converting the simplified NAND or NOR logic to a target technology, and simplifying the target technology where possible. The end result is an interconnection of primitives of the target technology in a language from which automated logic diagrams can be produced.

24 Claims, 33 Drawing Figures



OTHER PUBLICATIONS

Automated Exploration of the Design Space for Register-Transfer (RT) Systems: Barbacci Mario Roberto;

The Design and Analysis of an Automated Design Style Selector: Thomas Donald Earl, Jr.

Automation of Module Set Independent Register–Transfer Level Design: Edward Alfred Snow, III.

A new Look at Logic Synthesis; John A. Darringer; 17th DA Conference 1980.

Experiments in Logic Synthesis; John A. Darringer; IEEE ICCC 1980.

Methods used in an Automatic Logic Design Generator (Alert); Theodore D. Friedman; IEEE Transactions on Computers; vol. C–18, No. 7, Jul. 1969.

Register-Transfer Level Digital Design Automation: The Allocation Process; Louis Hafer; 15 DA Conference 1978.

The CMU Design Automation System; An Example of Automated Data Path Design; A. Parker et al.; 16 DA Conference 1979.

Lores–Logic Reorganization System; Shunichiro Nakamura et al.; 15 DA Conference 1978.

Translation of a DDL Digital System Specification to Boolean Equations; James R. Duley and Donald L. Dietmeyer, IEEE Trans. on Comp. vol. C–18. No. 14, '69.

DDL–A Digital System Design Language; James Robert Duley; Ph.D. 1967.

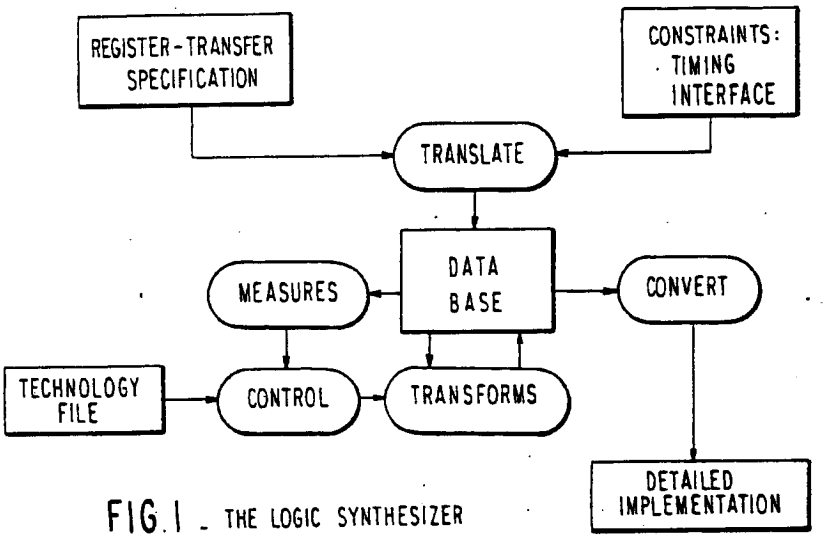


FIG. 1 - THE LOGIC SYNTHESIZER

FIG. 2

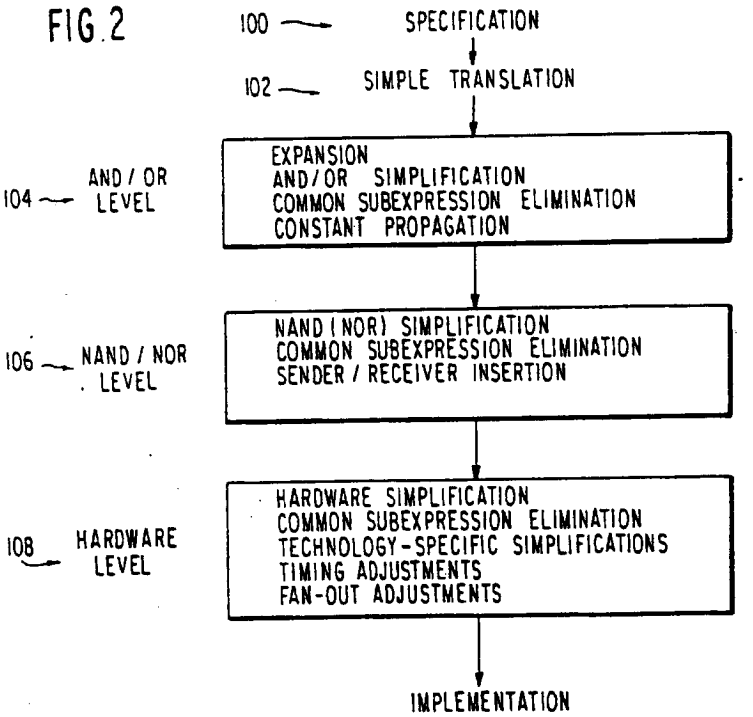


FIG.3a

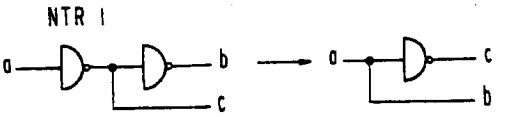


FIG.3b

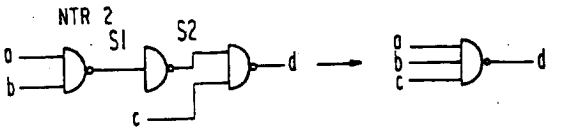


FIG.3c

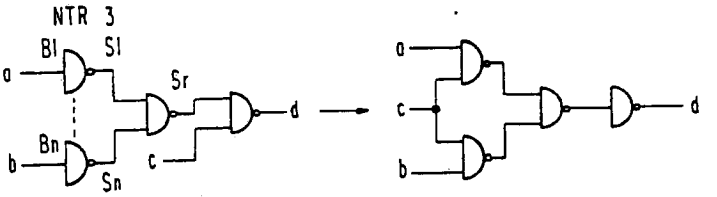


FIG.3d

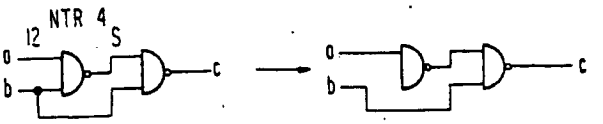


FIG.3e

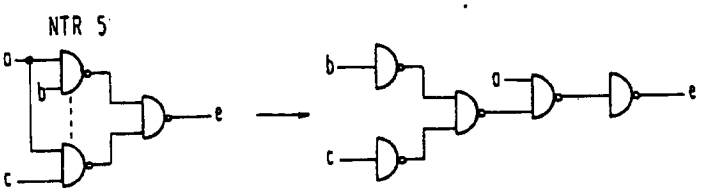


FIG.3f

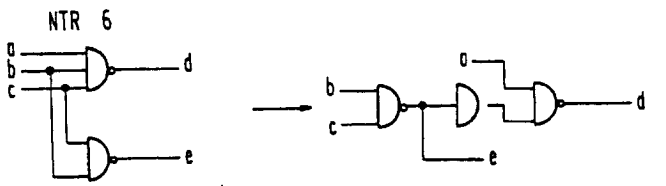


FIG.3g

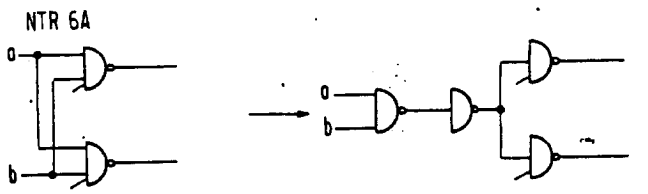


FIG.3h

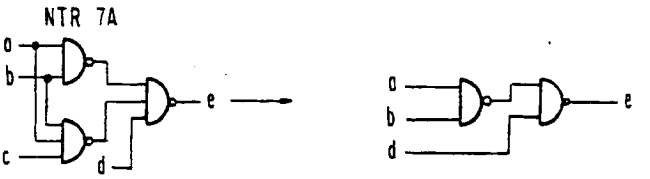


FIG.3i

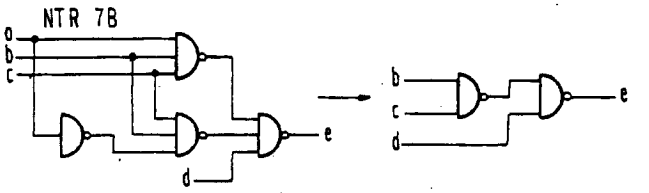


FIG.3j

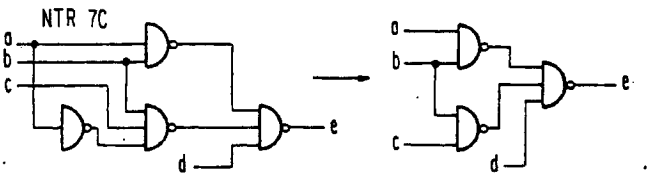


FIG.3k

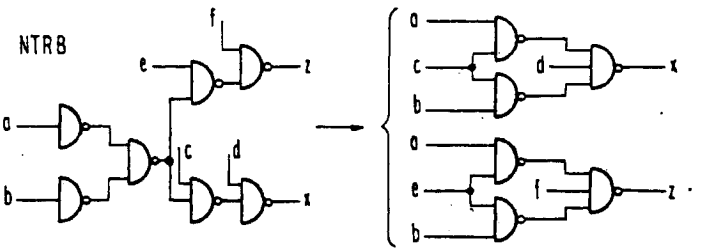


FIG.3l

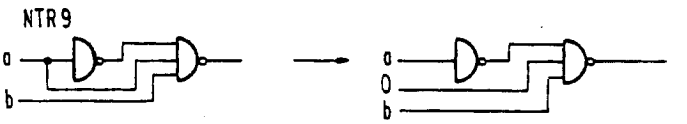


FIG.3m

FIG.3n

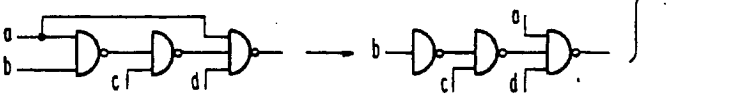


FIG.3o

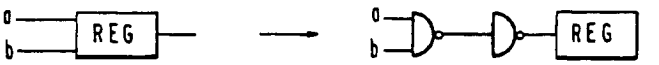


FIG.3p

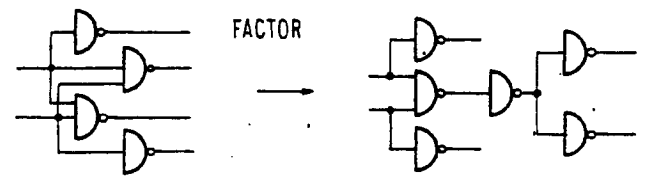


FIG. 4

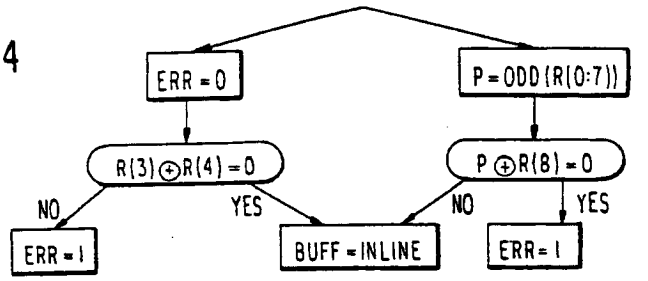


FIG. 5a
ELIMINATE COMMON TERMS

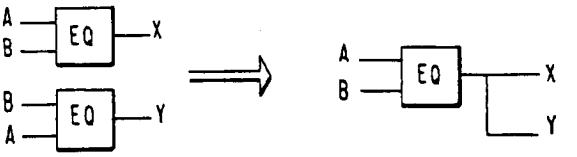


FIG. 5b
SIMPLIFY PARITY OPERATORS

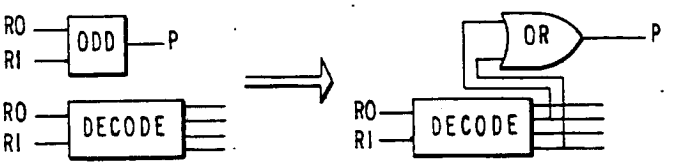


FIG. 6

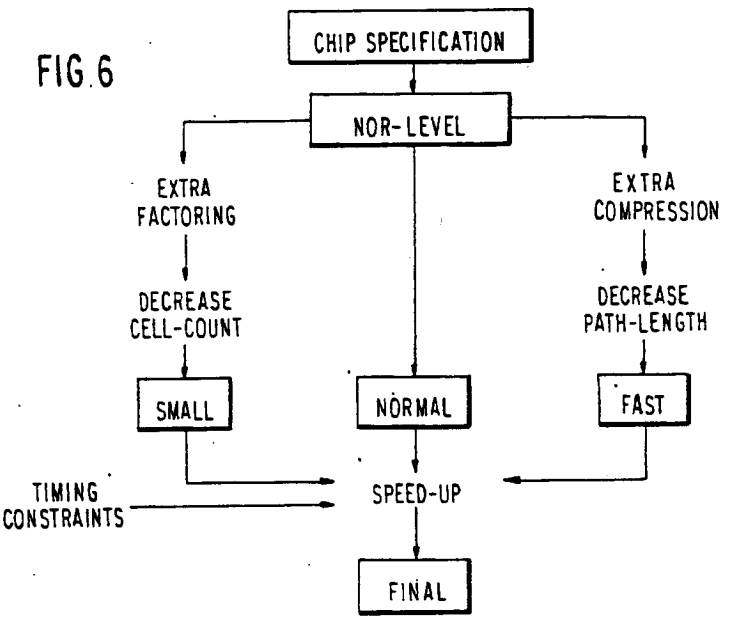


FIG.7a
MERGE RECEIVERS

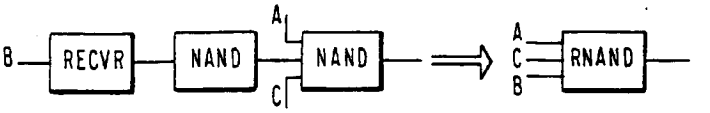


FIG.7b
INSERT DOTS

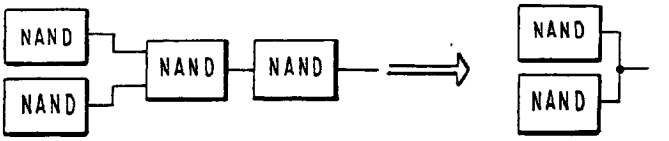


FIG.7c

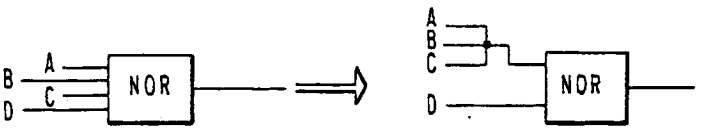


FIG.8a

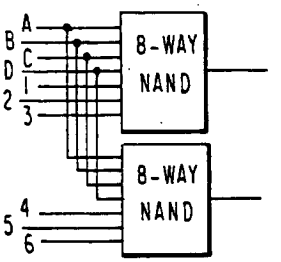


FIG.8b

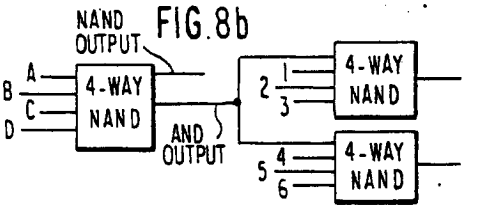


FIG.9a

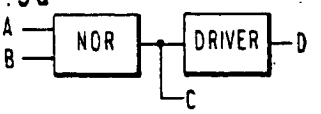
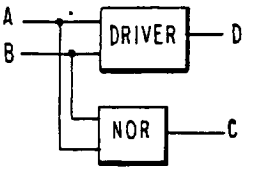


FIG.9b

FIG.9c



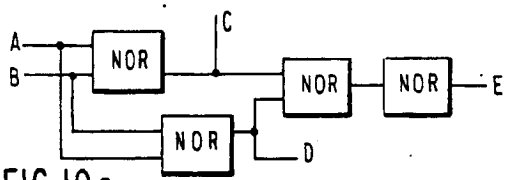


FIG.10a

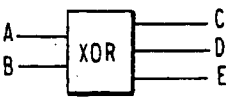


FIG.10b

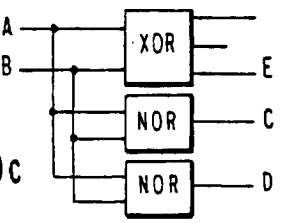


FIG.10c

LOGIC SYNTHESIZER

BACKGROUND OF THE INVENTION

This invention is directed to logic design, and more particularly to a method of automated logic design.

As the complexity of processors has increased, the task of processor logic design has become more difficult. The designer may begin by designing a flow chart or other register-transfer level description to describe the intended operation of the processor, and the processor operation is then simulated from this description in order to ensure that a processor operating in accordance with the flow chart will provide the desired results. A logic implementation is then designed to achieve the operation described in the flow chart, and the resulting logic diagram and original flow chart specification are compared to ensure consistency. Finally, a physical layout is designed in accordance with the logic diagram implementation.

The above process has become significantly more difficult and extraordinarily time consuming with the increasing complexity of the processors being designed. For example, each chip in the 3081 processor available from International Business Machines Corporation includes over 700 circuits capable of performing extremely complex functions. The flow chart specification of such a processor will be quite complex, and even a first attempt at a logic diagram implementation will require a substantial amount of time. Further, with increasing processor complexity, the competing interests of gate count and timing constraints become increasingly difficult to satisfy. More particularly, a typical timing constraint may be that a signal must be provided from the output of register A to the input of register B within some predetermined period of time, and the designer may first propose a logic arrangement intended to satisfy this timing constraint while using a minimal number of gates in the circuit path between registers A and B. After timing analysis, however, it may be discovered that the timing constraint has not been satisfied, and the designer must then revise the arrangement of logic between the registers A and B, e.g., by using a larger number of gates to improve the processing speed in that area. Several iterations of design may be required before a logic design is obtained which indeed satisfies all timing constraints with the minimum gate count, and it is therefore not uncommon for the logic design to be quite costly in terms of engineering time.

In view of the above, there has been significant recent activity in the field of automatic logic synthesis. Early work centered on developing algorithms for translating a boolean function into a minimum 2-level network of boolean primitives, and extensions were developed for handling limited circuit fan-in and alternative cost functions. However, because these algorithms employ 2-level minimization, the time required to implement these algorithms increases exponentially with the number of circuits. The use of such algorithms therefore becomes impractical in designing large processors.

Other efforts have attempted to raise the level of specification, e.g., by beginning with behavioral specifications and producing technology-independent implementations at the level of boolean equations. However, the results of such techniques were usually more expensive than manual implementations and did not take advantage of the target technology. For example, the

system described by T.D. Friedman et al, in "METHODS USED IN AN AUTOMATIC LOGIC DESIGN GENERATOR (ALERT)," IEEE Trans. on Computers, Vol. C-18, No. 7 pp. 593–614 (1969), produced an implementation for an IBM 1800 processor which required 160% more gates than the manual design for that processor. Several attempts have been made to produce more efficient logic and to give the designer more control over the implementation, e.g., as described by: H. Schorr, "Toward the Automatic Analysis and Synthesis of Digital Systems," Ph.D. Thesis, Princeton University, Princeton, NJ, 1962; C.K. Mestenyi, "Computer Design Language Simulation and Boolean Translation," *Technical Report* 68–72, Computer Science Department, University of Maryland, College Park, MD, 1968; F.J. Hill and G. R. Peterson, *Digital Systems: Hardware Organization and Control*, John Wiley & Sons, Inc., New York, 1973. However, this control has resulted in specification language constraints, so that the specification is at a fairly low level and in closer correspondence with the implementation. This necessarily decreases the advantage of an automated approach, bringing it closer to a system for logic entry rather than logic synthesis.

Several tools have been developed to support the early part of the design cycle, e.g., as described in: M. Barbacci, "Automated Exploration of the Design Space for Register Transfer Systems," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1973; D. E. Thomas, "The Design and Analysis of an Automated Design Style Selector," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1977; E. A. Snow, "Automation of Module Set Independent Register-Transfer Level Design," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1978; L. J. Hafer and A. C. Parker, "Register-Transfer Level Digital Design Automation: The Allocation Process," *Proceedings of the Fifteenth Design Automation Conference*, Las Vegas, NV, 1978, pp. 213–219; A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, and J. Kim, "The CMU Design Automation System - An Example of Automated Data Path Design," *Proceedings of the Sixteenth Design Automation Conference*, Las Vegas, NV. 1978, pp. 73–80. The technique described in the last-cited publication began with a functional description of a machine and produced an implementation in two technologies of the registers, register operators and their interconnections, but not the control logic to sequence the register transfers. For both TTL and CMOS implementations, however, the automated implementation required substantially more chip area than existing manual designs.

There has also been recent work in logic remapping, i.e., transforming existing implementations from one technology to another. S. Nakamura et al S. Nakamura, S. Murai, C. Tanaka, M. Terai, H. Fujiwara, and K. Kinoshita, "LORES-Logic Reorganization System," *Proceedings of the Fifteenth Design Automation Conference*, Las Vegas, NV, 1978, pp. 250–260; describe a system which will help a deslgner translate an existing small-or medium-scale integration implementation into large-scale integration. However, remapping usually involves one-to-one substitution of new technology primitives for old technology primitives, and this often fails to take advantage of simplification which may be available at a higher technology-independent level.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide an automated logic synthesis technique which overcomes the above-described drawbacks. It is a more particular object of the present invention to provide such an automated logic synthesis technique which is capable of operating at a relatively high speed while achieving end results comparable to those obtained by manual design. It is a still further object of this invention to provide such an automatic logic synthesis technique capable of achieving satisfactory results in a number of different technologies.

Briefly, these and other objects of the invention are achieved by a logic synthesis method in which a register-transfer level flowchart specification is translated in a straightforward manner into a simple AND/OR logic implementation. After expanding the logic implementation to elementary representation and then applying textbook simplifications, the simplified AND/OR implementation is translated to a NAND or NOR implementation, depending on the target technology. The NAND or NOR implementation is then simplified by applying a sequence of simplification transformations which have been found by the present inventors to achieve satisfactory results, with the transformation sequence being modified to achieve "normal," "fast" or "small" logic designs. After simplification at the NAND/NOR level, the logic implementation is then translated to the target technology and further simplified. The result is an interconnection of the primitives of the target technology in a language from which automated logic diagrams can be produced in a known manner, and which can be submitted to existing programs for automated placement and wiring and chip fabrication.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be more clearly understood from the following description in conjunction with the accompanying drawings, wherein:

FIG. 1 is a conceptual diagram of the logic synthesis technique according to the present invention;

FIG. 2 is a chart illustrating the multiple levels of simplification in the logic synthesis technique according to the present invention;

FIGS. 3(*a*)–3(*p*) illustrate simplifying transformations at the NAND/NOR level;

FIG. 4 is a simple illustration of a portion of a flowchart specification from which the present invention begins;

FIGS. 5(*a*)–(*b*) illustrate simplifications which may be performed at the AND/OR level;

FIG. 6 is a diagram illustrating the different scenarios of simplification at the NAND/NOR level;

FIG. 7(*a*)–7(*c*) illustrate examples of simplification at the hardware level; and

FIGS. 8(*a*)–8(*b*), 9(*a*)–(*c*) and 10(*a*)–10(*c*) illustrate further examples of technology-specific hardware simplifications.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The logic synthesis method according to the present invention is generally illustrated in FIG. 1. Previous publications describing some aspects of the system according to this invention, all of which are incorporated herein by reference, are: J. A Darringer and W. H. Joyner, "A New Look at Logic Synthesis," *Proceedings of the Seventeeth Design Automation Conference* Minneapolis, MN, 1980, pp. 543–549; J.A. Darringer, W. H. Joyner, L. Berman, and L. Trevillyan, "Experiments in Logic Synthesis," *Proceedings of the IEEE International Conference on Circuits and Computers ICCC80*, Port Chester, NY, 1980, pp. 234–237A; J. A. Darringer, W. H. Joyner, C. L. Berman and L. Trevillyan, "Logic Synthesis Through Local Transformations," *IBM Journal of Research and Development, Vol.* 25 No. 4, July 1981.

The present invention is an automatic replacement for part of the manual design process. It operates on a logic design at three levels of abstraction. It begins with an initial implementation generated in a straightforward manner from the specification. The implementation can be simplified at this level, and then moved to the next level. This simplification is accomplished by transformations, either locally or globally, to achieve the simplification or refinement. By being able to operate on the implementation at several levels, the system can often make a small change at one level that will cause a larger simplification at a lower level. By using function-preserving transformations, it is ensured that in all cases the implementation produced will be functionally equivalent to the specified behavior. The inputs to the system illustrated in FIG. 1 are a description, in a register-transfer level, flow chart-control language, of logic functions to be implemented on a chip in a specified master slice technology, together with the interface constraints and a technology file which characterizes the target technology. The output of the system is a detailed interconnection of the primitives of the target technology in a language from which automated logic diagrams (ALD's) may be produced and which can be submitted to existing programs for automated placement and wiring and chip fabrication. The output implementation is in terms of the target technology and satisfies technology-specific constraints. Some timing or other physical problems may not be detectable before placement and wiring, and in such cases the synthesis process is repeated with a revised specification or modified constraints until an acceptable implementation is achieved.

The method according to this invention comprises PL/I programs operating on a representation of the logic in a data management system. The data management system is preferably that described by F. E. Allen et al, "THE EXPERIMENTAL COMPILING SYSTEM," IBM Journal of Research and Development, Volume 24 (1980), pages 695–715. The logic synthesis data base uses a single organization component referred to as a "box," with each box having input and output terminals which are connected by wires to other boxes. Each box also is designated by a type, which may be a primitive or may reference a definition in terms of other boxes. Thus, a hierarchy of boxes can be used, and an instance of a high-level box such as a parity box can be treated as a single box or expanded into its next-level implementation when that is desirable.

The logic synthesis data base is made of two groups of tables. The first group describes the technology being used, and is created from a technology file containing, for each box type, information such as name, function and number and names of input and output pins. These data are created in batch mode and read during initialization of the interactive system.

The second group of tables contains the representation of the logic created by the system. This group consists of a box table, a signal table and a set of auxiliary tables which describe the relationship between the boxes and the signals. There is some intentional redundancy in the data, i.e., each box has a complete list of input and output signals and each signal has a source and a list of sinks. Every box table entry contains type information which provides a link to the technology group, thus allowing programs to obtain technology information about a specific box.

Using the system generally illustrated in FIG. 1, a synthesis process according to the present invention may follow the sequence of steps shown in FIG. 2. FIG. 2 illustrates the three essential levels of description used in the method of the present invention: the initial AND/OR level 104, a NAND or NOR level 106 (depending on the target technology), and a hardware level 108 in which the types of the boxes are books or primitives of the target technology. At every level, the implementation is a network of boxes connected by signals. The purpose of this type of implementation is to find a set of transformations and a sequence of applying these transformations such that the original functional specification could be transformed by a sequence of small steps into an acceptable implementation.

As pointed out above, the process of this invention begins at step 100 with a register-transfer level description e.g. of the type shown in FIG. 4. The description consists of two parts: a specification of the inputs, outputs and latches of the chip to be synthesized; and a flowchart-like specification of control, describing for a single clock cycle of the machine how the chip outputs and latches are set according to the values of the chip inputs and previous values of the latches. At step 102 in FIG. 2, the register-transfer level description undergoes a simple translation to an initial implementation of AND/OR logic. This AND/OR level is produced by merely replacing specification language constructs with their equivalent AND/OR implementations in a well known manner, e.g., as described in J. R. Duley, "DDL - A Digital Design Language," Ph.D. Thesis, University of Wisconsin, Madison, WI, 1968; or J. A. Darringer, "The Description, Simulations and Automatic Implementation of Digital Computer Processors," Ph. D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, 1969. At this first level 104 in FIG. 2, the logic begins in the form of an interconnection of boxes designated by types representing the operations which the perform, e.g., AND, OR, NOT, PARITY, EQ, XOR, DECODE, REGISTER (generic latch), SENDER, RCVR. At step 104 in FIG. 2, the initial AND/OR implementation is first expanded by taking all operators more complex than AND, OR or NOT and replacing these more complex operators with combinations of AND, OR, and NOT. Beginning with this expanded AND/OR logic, simplification is achieved by invoking PL/I program transformations which search for patterns of interconnected primitives and replace them by functionally equivalent patterns which are simpler in that they use fewer instances of operators, fewer connections, etc. The transformations at the AND/OR level 104 are local, textbook simplifications of boolean expressions, most of these simplifications reducing the number of boxes but not producing a normal form. Examples of simplifications are shown in FIGS. 5(*a*) and 5(*b*). Some of these transformations are similar to optimizing compiler techniques, e.g., constant propagation (moving "0" or "1" signals through logic blocks), common term elimination (combining blocks which compute the same function), combining nested associativecommutative operators, eliminating single input AND's and OR's, etc. Further examples of transformations used are as follows:

NOT(NOT(a))→a
AND(a, NOT(a))→0
OR(a, NOT(a))→1
OR(a, AND(NOT(a),b)→OR(a,b)
XOR(PARITY(a_1, . . ., a_n), b)→
PARITY(a_1, . . ., a_n, b)
AND (a, 1)→a
OR(a, 1)→1

These transformations may leave fragments of logic disconnected, and this can be cleaned up in a manner similar to the way in which compilers perform dead-code elimination.

After simplification at the AND/OR level 104, the simplified AND/OR implementation is transformed into a NAND or NOR implementation. Whereas AND/OR logic requires the use of multiple different operators in a logic design, NAND or NOR logic requires fewer operators, i.e., in a NAND logic design all logical functions can be implemented using some combination of only NAND gates. Whether a NAND or NOR implementation is produced is dependent upon the primitives available in the target technology. However, the NAND or NOR description is not technology-specific, in that there are no fan-in or fan-out restrictions. (Fan-in refers to the number of signals coming into a box, and fan-out refers to the number of sinks or destinations of a signal.) The transition to these primitives is accomplished naively by local transformations and may introduce unnecessary double NANDs or NORs which will be eliminated later. Also at this point, the chip interface information is used to place generic, i.e., not technology-specific, senders and receivers on the chip inputs and primary outputs, and to insert inverters where necessary to ensure the correct signal polarities. Techniques for accomplishing this transformation are well-known and need not be described here in detail.

At step 106 in FIG. 2, simplifying transformations are applied to each signal in the network. The NAND and NOR transforms are more difficult, and extensive experiments by the present inventors at the NAND/NOR level have resulted in a sequence or "scenario" of transformations which will produce acceptable results. The transformations are local in that they replace a small subgraph of the network (usually five or fewer boxes) with another subgraph which is functionally equivalent but simpler according to some measure. These transformations attempt to reduce the number of boxes of the implementation without increasing the number of connections. To accomplish this, the transformations must check the fan-out of the various signals involved, since this will affect the number of boxes and signals actually removed. Some of the transformations attempt to remove reconvergent fan-out which contributes to untestable stuck faults.

Some of the transformations are applied throughout the network in a number of iterations, preferably until no more transformations apply. FIGS. 3(*a*)–3(*n*) illustrate the NAND transformations NTR1 thru NTR10 used in one embodiment of this invention, and the NOR transformations would be identical except for the operator. Each transformation has an associated condition that determines if the replacement will simplify the

implementation by reducing boxes or connections. These conditions depend on the fan-out of the intermediate signals and on whether the target technology is assumed to have dual-rail output.

Experiments with the NAND/NOR level transformations have resulted in a normal sequence or "scenario," of transformations which have produced acceptable results. A "fast" scenario was developed which resulted in shorter path lengths, and a "small" scenario was also developed to obtain smaller designs. These are generally indicated in FIG. 6. In the preferred embodiment of this invention, the sequence of steps in the normal NAND/NOR scenario would be as follows:

APPLY GENNOR: (or APPLY GENNAND);
UNTIL NOCHANGE APPLY NTR1, NTR2, CLEANUP, NTR3, NTR4, NTR10, CLEANUP, NTR7, NTR9, PROPCON, CLEANUP, CTE, CLEANUP;
FANIN 4;
APPLY NTR6A, FACTORN, NTR6A, CLEANUP;
APPLY NTR10, CLEANUP, NTR7 (NOINCREASE), NTR9, PROPCON, CLEANUP:
APPLY CTE, CLEANUP; FANIN 8:
APPLY NFANIN, NTR11, CLEANUP;

The GENNOR or GENNAND transformations merely transform the AND/OR implementation into either NAND or NOR logic in accordance with the target technology. This type of transformation is well understood in the art and need not be described in detail here.

NTR1 in FIG. 3(*a*) removes double inverters and always applies, since it is always considered desirable to reduce the number of cells, and because this transformation does not increase connects or path lengths. This transform, and others, may in some instances increase fan out, but the fan out can be reduced, if necessary, at a later point.

NTR2 in FIG. 3(*b*) applies only if s_1 has no fan out and s_2 fans out only to primitives, i.e., either NANDs or NORs. This transform will not apply if it will result in an increase in the number of connects. For example, in the transformation illustrated in FIG. 3(*b*), gates 10 and 12 are eliminated and their corresponding input and output connections are also eliminated. However, if s_2 fans out to four NANDs, it would be necessary to apply the NTR2 transformation to each one, resulting in an increase in the number of connects.

NTR3 in FIG. 3(*c*) applies only if none of the gate outputs s_i fans out, s_r does not fan out, and no gate B_i exceeds the fan-in threshold for a single-cell book. This helps set up later dotting.

NTR4 in FIG. 3(*d*) removes redundancy locally. Redundancy is a property of a combinational logic circuit, and is present when the network contains a signal that can be set to a constant value without changing the function of that network. NTR4 also replicates logic if the output s of gate 12 fans out.

NTR6A in FIG. 3(*g*) sets up dotting and is only run if dotting is allowed in the target technology.

NTR7 eliminates some forms of redundant connections. This transform will replicate boxes, if necessary, unless the parameter NOINCREASE is specified. NTR7 actually comprises three transforms illustrated in FIGS. 3(*h*)–3(*j*), all of which are run each time NTR7 is called for in the above program.

NTR9 in FIG. 3(*i*) handles cases where a signal and its negation both go to a NOR or NAND gate. The "0" input to gate 14 will be a "1" for the equivalent NOR transformation. This transform should be followed by PROPCON, described below.

NTR10 includes two different transforms illustrated in FIGS. 3(*m*) and 3(*n*), both of which are run each time NTR10 is called for. The NTR1O transform is run only if the outputs of gates 18 and 20 and FIG. 3(*n*) do not fan out.

NTRll in FIG. 3(*o*) makes all generic registers (considered to have the OR function) have a fan-in of 1 by preceding each register with an appropriate number of primitives.

PROPCON, CLEANUP and CTE are analogous to the compiler operations of constant propagation elimination, dead-code elimination and common sub-expression elimination, respectively. Common sub-expression elimination, or common term elimination, refers to locating boxes which produce the same logic value, eliminating one box, and sharing the output of the other box.

FANIN 4 does not in itself perform any transformation but instead sets a variable known as "FANIN" to a value of 4.

FACTORN examines only boxes exceeding the FANIN limitations specified by the variable FANIN. It then applies the transformation of FIG. 3(*p*). This transformation will not reduce all boxes to below the specified FANIN limit, but only those boxes to which it applies by finding common sinks.

NFANIN corrects the fanin to the specified limit by building fanin trees which it constructs to have the fewest boxes and then to lengthen as few paths as possible.

In a NOCHANGE loop, the transformations are repeatedly run in their specified order until no further change in the logic occurs. In general, the order of the transformations and their inclusion in the NOCHANGE loop is such that suceeding transformations are invoked when preceding transformations can cause them to apply. For example, in the first loop, the sequence beginning with NTR9 is used to remove gates having complementary inputs. Since this can produce constant zeros or ones, constant propagation (PROPCON), removal of unconnected boxes (CLEANUP), common term elimination (CTE), and then more CLEANUP (to deal with now-unconnected common terms) must be run. On the other hand, after fan-in correction by factoring and NFANIN, some transformations should not be run, because they may destroy the fan-in limits already enforced.

In looking again at the program above, it can be seen that certain sequences of functions are performed, with some functions comprising a plurality of transformations. More particularly, with regard to the first NOCHANGE loop, transformations NTR1, NTR2, CLEANUP, NTR3 operate to reduce logic depth, i.e., number of levels of logic from input to output, with NTR1 reducing logic depth from two levels to one and NTR2 reducing logic depth from three levels to one. NTR3 at first glance appears to provide no depth reduction, since it transforms three levels of logic to three levels of logic. However, in some instances the last level, gate 11, can be subsequently eliminated, so that NTR3 is often useful in reducing logic depth.

Reducing logic depth, i.e., .compressing the logic into fewer levels, will increase the chance of detecting redundancy. Thus, NTR4, NTR10, CLEANUP, NTR7, NTR9, PROPCON, CLEANUP applied to remove redundancy.

After removing redundancy, a common terms elimination sequence CTE, CLEANUP is run.

After the NOCHANGE loop has finished running, transformations are applied to introduce dot patterns and to reduce fan-in to a specific level. This is accomplished by the step FANIN 4 which sets the fan-in limit to a value of 4, followed by the sequence NTR6A, FACTORN, NTR6A, CLEANUP, which serves to reduce fan-in at the expense of logic depth.

Once again, the introduction of dot patterns and the factoring to reduce fan-in may result in redundancy, so that the redundancy removal sequence NTR10, CLEANUP, NTR7, NTR9, PROPCON, CLEANUP is applied.

Common terms are then eliminated by running CTE, CLEANUP.

Finally, the logic must be adjusted to the maximum fan-in value permitted by the target technology, e.g., a fan-in value of 8. This is achieved by applying FANIN 8 to set the fanin value at 8 followed by NFANIN, CLEANUP.

As should now be appreciated, the above program can be functionally represented as follows:

A. LOGIC DEPTH REDUCTION LOOP
  A1. REDUCE LOGIC DEPTH
  A2. REMOVE REDUNDANCY
  A3. ELIMINATE COMMON TERMS
B. INTRODUCE DOT PATTERNS AND FACTOR TO REDUCE FANIN TO SPECIFIC LEVEL
C. REMOVE REDUNDANCY
D. ELIMINATE COMMON TERMS
E. ADJUST LOGIC TO MAXIMUM PERMITTED FANIN

The operations subsequent to the logic depth reduction loop may tend to expand the logic depth, so that the above process can generally be seen as a compression stage followed by an expansion stage. While it may be theoretically possible to obtain maximum logic depth reduction through two-level boolean minimization, this would compress the logic so far that re-expansion to take advantage of other simplifying transformations, e.g., at the subsequent hardware simplification, would be much more difficult. Thus, the logic compression transforms have been found particularly suitable.

The program set forth above concerns a normal scenario, and the "fast" and "small" scenarios can be obtained by modifying the above program as follows: for the small scenario, the following additional NOCHANGE loop is inserted after the NOCHANGE loop in the normal scenario:

UNTIL NOCHANGE APPLY NTR6, NTR5, NTR1, NTR2, CLEANUP, NTR3, NTR4, NTR10, CLEANUP, NTR7, NTR9, PROPCON CLEANUP, CTE, CLEANUP;

NTR5 in FIG. 3(*e*) applies only if the number of cells does not increase, and NTR6 in FIG. 3(*f*) applies only if the number of cells is decreased. Inspection of NTR5 and NTR6 shows that they can increase path length, and they are consequently only used in the small scenario. The other transformations in the added loop are provided to act on any changes which may result from NTR5 and NTR6. For example, NTR5 and NTR6 can produce double inverters, so the sequence beginning with NTR1 is run. NTR1 eliminates double inverters, and can introduce situations where other transforms apply.

Examination of the second NOCHANGE loop set forth above will reveal that the loop includes a first sequence NTR6, NTR5 for reducing the cell count by increasing the logic depth. The sequence NTR1, NTR2, CLEANUP, NTR3 is then applied to mitigate the logic depth reduction by taking advantage of transforms made available by NTR6, NTR5. After this logic depth reduction sequence, the redundancy removal and common term elimination sequence are applied in the first NOCHANGE loop.

Thus, the program for the "small" scenario can be written:

A. LOGIC DEPTH REDUCTION LOOP
  A1. REDUCE LOGIC DEPTH
  A2. REMOVE REDUNDANCY
  A3. ELIMINATE COMMON TERMS
  A'. CELL COUNT REDUCTION LOOP
  A1'. REDUCE CELL COUNT
  A2'. REDUCE LOGIC DEPTH
  A3'. REMOVE REDUNDANCY
  A4'. ELIMINATE COMMON TERMS
B. INTRODUCE DOT PATTERNS AND FACTOR TO REDUCE FANIN TO SPECIFIC LEVEL
C. REMOVE REDUNDANCY
D. ELIMINATE COMMON TERMS
E. ADJUST LOGIC TO MAXIMUM PERMITTED FANIN

While the "small" scenario is designed to emphasize minimization of gate count, the "fast" scenario is designed to emphasize shorter path lengths, sometimes at the expense of gate count. Path length refers to the delay along a path from a signal's source to one of its destinations. Usually, path lengths are measured from registers or primary chip inputs to registers or primary chip outputs. The result can be the number of boxes in the path or the estimated delay of that path in nanoseconds.

The fast scenario inserts a call to NTR8 as the last step run in the first NOCHANGE loop. Immediately thereafter, FANIN is set to a value of 8 rather than 4, and NTR11 is omitted from the last line of the program. The significance of these changes to the program is as follows:

NTR8 in FIG. 3(*k*) is used in the fast scenario because it shortens paths. This may sometimes be at the expense of cells, however, since some of the boxes shown in FIG. 3(*k*) may have to be replicated. The factoring to a fanin of 8, also produces shorter paths, but may increase the cell count, e.g., in a dual-rail technology in which a 4-way NOR/OR required one cell and an 8-way required two cells. This will be explained in more detail with reference to FIGS. 8(*a*) and 8(*b*).

In a particular technology, there may be a number of different primitives or "books" having different fan-in capabilities, and different books may include different numbers of cells. For example, an 8-way NAND gate may use two cells while a 4-way NAND gate may use one cell. If 8-way NAND gates are used, e.g., to combine ten different inputs in two combinations with four inputs common to each combination, the result may be as shown in FIG. 8(*a*). Each book would receive seven inputs, and a total of four cells would be used.

If fan-in is limited to a value of 4, the same logic could be implemented as shown in FIG. 8(*b*) using three 4-way books. Although the number of books has increased, each book includes only one cell, so that the cell count decreases from four to three. However, the

cell count decrease is at the expense of increasing the logic depth by one level.

In the "normal" or "small" scenarios, it is worthwhile to set the fan-in value to 4 and to factor in an attempt to take advantage of the cell reduction which may be realized by using the smaller books. In the "fast" scenario, however, the increase in logic depth accompanying the use of the smaller books is unacceptable, and the fanin is instead set to the maximum allowable fan-in after the NOCHANGE loop.

Thus, the simplification program for the "fast" scenario can be functionally described as follows:

LOGIC DEPTH REDUCTION LOOP
A1. REDUCE LOGIC DEPTH
A2. REMOVE REDUNDANCY
A3. ELIMINATE COMMON TERMS
A4. REDUCE LOGIC DEPTH WHILE INCREASING CELL COUNT
B. INTRODUCE DOT PATTERNS AND FACTOR TO REDUCE FANIN TO MAXIMUM ALLOWED BY TECHNOLOGY
C. REMOVE REDUNDANCY
D. ELIMINATE COMMON TERMS
E. ADJUST LEVEL TO MAXIMUM PERMITTED FANIN

The search strategy for the above transformations is to scarch the interconnected boxes of the date-base in sequence, looking for a pattern to which the transform may apply. The search is done for each transform in an efficient way, e.g., NTR2 searches the entire logic design for a one-input inverter, since this is faster than examining each multi-way NAND or NOR to determine if an inverter precedes or follows it.

After the simplification sequence described above, transformations are applied to the logic to map the NAND or NOR implementation to the target technology, simplify the technology-specific implementation, and enforoe technology-specific restrictions. This is performed at level **108** in FIG. 2. The transformations applied at level **108** may be generally described as follows, although their exact implementation will depend on the target technology

Technology-specific transforms may preferably be applied in the following order: first, generic NAND/-NOR gates are mapped to their counterparts in the target technology. If the fan-in of a gate is too high and there is no corresponding primitive in the technology, a tree of primitives must be built to produce the same logical function. REG's, the generic latches, are mapped to the technology-specific latches. In general, the technology-specific latches have a limited number of pins for data values. If more data values are gated into the latch than can be accomodated, extra "ports" must be connected to the latch in a manner prescribed for the technology. SENDER's and RECEIVER's are mapped to their technology-specific counter parts.

Second, if the target technology is dual-rail, dual-rail books are introduced. With both positive and negative phases available from each gate, all inverters (except those on chip inputs) are removed and their output signals connected to the opposite phase of the source of their input signals. Third, technology-specific "tricks" are introduced, e.g., special books, drivers, receivers, etc., which were not known at the time of the generic transformation. These implement certain functions, such as XOR, combinations of driver and logic functions, combinations of receiver and logic funtions, combinations of latch and receiver, etc., using fewer cells than the primitive NAND or NOR implementation. The pattern of technology-specific NAND's or NOR's is searched for and replaced by the appropriate block. In FIG. 7(*a*), three cells can be replaced with a single NAND in the target technology having a built-in receiver.

If dots, i.e., wired AND's or OR's, are allowed in the target technology, patterns implementing +AND or +OR functions are located. If the inputs of these patterns have no fan-out, the pattern is replaced by a dot, e.g., as shown in FIG. 7(*b*). Dots can also be introduced to reduce fan-in as shown in FIG. 7(*c*). After dots are introduced, more special books may be present and are searched for again.

Next, fan-out is adjusted to meet constraints. Fanout limits are specified for technology-specific box types and output pins of those boxes. Fan-out is brought within these limits by replicating the violating box and distributing some of its fan-out to the copy (parallel repowering) or driving some of the fan-out with a repowering +OR or +AND function in front of the violating box (serial repowering). Additional dual-rail books are added after fan-out adjustment, but not so as to violate fan-out constraints.

Next, clock signals are introduced as chip inputs and distributed to the latches of the chip according to technology-specific requirements for clock distribution. Depending on the technology, this requires clock balancing and introduction of special clock drivers.

Next, path lengths back from latch-to-latch and from chip output to chip input are analyzed. Long paths are first shortened by rearranging fan-in and fan-out repowering, introducing dots (even at the cost of cell count), "undoing" factoring transformations performed at a higher level and introducing high-power books. Short paths are then padded to meet minimum path length requirements.

The fan-out adjustment is then repeated to correct fan-out violations which may have resulted from the path length correction.

Finally, scan-in and scan-out pins are introduced and the latches are linked together in an LSSD scan ring. A chip-in-place test and/or inhibit signals are introduced for chip outputs where required. Since this may introduce fan-out violations, fan-out adjustment is repeated.

An example of a hardware conversion and simplification program for a NOR technology following the above-described sequence may be as follows:

APPLY GENHW, CLEANUP;
APPLY DUAL (NO LIMIT), CLEANUP;
APPLY OPTDRIVE (SMALL OR FAST), CLEANUP,
OPTXOR (SMALL OR FAST), CLEANUP;
APPLY GENDOT;
APPLY OPTDRIVE (SMALL OR FAST), CLEANUP, OPTXOR (SMALL OR FAST), CLEANUP;
APPLY FANOUT, DUAL, CLEANUP;
APPLY CLOCK;
APPLY TIMINE;
APPLY FANOUT, DUAL, CLEANUP
APPLY SCANP, FANOUT;

GENHW maps generic gates to hardware primitives. Since fan-in has been adjusted at the end of the NAND/NOR simplification, much of this step is merely one-to-one mapping.

DUAL removes necessary inverters in dual-rail technologies by absorbing the inverters into other gates

which already have positive and negative phases available. This transform will normally be applied so as to exceed the fan-out limit. However, with the NOLIMIT option this transform will always apply.

OPTDRIVE takes advantage of a technology-specific book available, i.e., a driver book with built-in NOR capability. As shown in FIG. 9(*a*), the logic design may at this point include a NOR gate with a branched output with one branch going to a driver. Since both functions can be served by a single book in the target technology, the arrangement of FIG. 9(*b*) can be substituted. However, while this may be desirable in "normal" and "small" scenarios, there is a sacrifice in speed. Thus, for a "fast" scenario, the transformation is to the arrangement shown in FIG. 9(*c*) which provides for "parallel" operation and therefore higher speed at the expense of cell count.

OPTXOR takes advantage of a further technologyspecific book, i.e., the XOR book. This transformation searches for a pattern of NOR gates providing the XOR function, e.g., as shown in FIG. 10(*a*), and substitutes the XOR book as shown in FIG. 10(*b*). Again, however, the transformation to FIG. 10(*c*) is employed in the "fast" scenario.

GENDOT introduces dotting in such a manner as to both eliminate gates and reduce fan-in. E.g., the transformation shown in FIG. 7(*b*) will eliminate gates 15 and 16 while the transformation shown in FIG. 7(*c*) will not eliminate gate 17 but will reduce the fan-in to that gate. This may save cells by permitting the use of a smaller book in the target technology and by allowing other transforms to apply. Since GENDOT changes the logic, OPTDRIVE and OPTXOR are applied again to search for more special books which may now exist.

FANOUT is applied to reduce the fan-out to the allowed limit. Note that the first half of the above hardware level simplification program is run without regard to fan-out limitations, as even the DUAL transform is applied with its NOLIMIT option. The various transformations may have caused fan-out violations which should be corrected by applying FANOUT in the manner discussed above. DUAL is then applied again, but this time so as not to violate fan-out constraints.

CLOCK is applied to distribute clock signals according to technology specific requirements in a manner known in the art.

TIMING is applied to correct path lengths by rearranging fan-in and fan-out trees, introducing more dots and chaning power levels to shorten long path lengths, and inserting pad logic to lengthen the short paths, as necessary. After TIMING is applied, fan-out adjustment is again performed to correct any violations which may have resulted from the timing correction, and DUAL is again run, within fan-out constraints, to take advantage of changes made during fan-out adjustment.

Finally, SCANP is applied to link the registers in a LSSD scan path. Fan-out is again adjusted to correct any violations which may have resulted from SCANP.

The logic synthesis system of this invention employs three different levels of simplification between the original specification and the final implementation: high level simplifications, NAND/NOR level simplifications and technology specific simplifications. Several of the transforms at the three different levels are analogous, differing only in the types of boxes to which they apply, so that simplifications not made at one level would be caught later. This may appear redundant, but the application of transforms as early as possible reduces the size of the implementation and helps prevent a greater explosion in size when, e.g., conversion to NANDs takes place.

A significant advantage of the present invention resides in its adaptibility to more than one technology, requiring modifications to only a part of the system and leaving the technology-independent portions intact. This makes the synthesis process according to the present invention useful in synthesizing logic in a number of different technologies, and in fact facilitates the remapping from one technology to another in an efficient manner. Rather than merely mapping hardware primitives one-to-one from one technology to another, a first technology implementation is abstracted to a technology-independent level, e.g., from a TTL chip implementation to a NAND level implementation with generic registers, drivers and receivers. The NAND implementation can be mapped to a NOR level implementation in a straight-forward manner, with the NOR level simplification being performed in the manner described above with reference to level **106** in FIG. 2. The hardware mapping and simplification can then be performed in the manner described with reference to level **108** in FIG. 2. This enables the remapping to take advantage of simplifications which may be available at the NOR level.

Some of the work described in the earlier-cited publications concerned a synthesis process beginning with a behavioral description and producing technology-independent implementations of boolean equations. These processes did not take advantage of the target technology. Other work has centered on the synthesis of the data-flow portion of a machine, synthesis from a high-level behavioral description to a register-transfer description, and implementation of control logic in microcode or programmable logic arrays. In contrast, the present invention provides the following significant features:

First, the present invention uses local transformations at several levels of description, passing through technologyindependent levels of description to a technology-specific description. This enhances the simplification while also facilitating the re-implementation of a design in a different technology.

Second, the specific sequences of simplifying transformations and the conditions associated with them have been found to provide acceptable results in normal, fast and small scenarios, thus making automated logic synthesis practical.

Further, timing, driver and other interface constraints are used at the hardware level to generate logic meeting these requirements.

Still further, the automated logic synthesis process according to the present invention greatly facilitates timing analysis and correction of the design to remove path length problems.

What is claimed is:

1. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating, via said computer, a first logic circuit design in a first logic system in accordance with said description;

simplifying said first logic design via said computer;

converting, via said computer, said simplified first logic design to a second logic design in a second logic system requiring fewer different logic operators than said first logic system, said second logic system comprising a plurality of interconnected cells and performing equivalent functions;

simplifying, via said computer, said second logic design, said step of simplifying said second logic design comprising the steps of: applying a depth reduction sequence of logic transformations for reducing the depth of said second logic design; and subsequently applying a size reduction sequence of logic transformations for reducing the size while possibly increasing the depth of said second logic design; and

converting, via said computer, said simplified second logic design to said desired technology.

2. A method as defined in claim 1, wherein said step of applying said depth reduction sequence of logic transformations comprises: applying a first logic transformation set for reducing logic depth; applying a second logic transformation set for reducing redundancy; and applying a third logic transformation set for eliminating common terms.

3. A method as defined in claim 2, wherein said step of applying said size reduction sequence of logic transformations comprises: applying a fourth logic transformation set for reducing logic size while possibly increasing logic depth; applying a fifth logic transformation set for reducing redundancy; and applying a sixth logic transformation set for eliminating common terms.

4. A method as defined in claim 3, wherein said second and fifth logic transformation sets include at least one common logic transformation. said common logic transformation being applied in said depth reduction sequence regardless of whether said common logic transformation will increase the number of cells in said second logic design and being applied in said size reduction sequence only if it will not result in an increase in said number of cells.

5. A method as defined in claim 2, wherein said step of applying said depth reduction sequence of logic transformations further comprises applying a fourth logic transformation set (e.g.. NTR8), following said third logic transformation set, for further reducing said logic depth while increasing the number of cells in said second logic design.

6. A method as defined in claim 3, wherein said converting step comprises converting said simplified second logic design to a hardware logic design in said desired technology represented as a plurality of hardware primitives. said method further comprising the step of simplifying said hardware design, said hardware simplifying step comprising:

applying a first hardware transformation set for substituting technology-specific books for predetermined patterns of said hardware primitive;

dotting signal lines to decrease the number of components in said hardware logic design, and to decrease fan-in in some portions of said hardware logic design even if the number of components in said portions is not decreased;

applying said first hardware transformation set;

correcting fan-out in said hardware logic design to a desired value;

adjusting path lengths in said hardware logic design; and

correcting fan-out to said desired value.

7. A method as defined in claim 1, wherein said step of applying said size reduction sequence of logic transformations comprises applying a first logic transformation (e.g., FACTORN) for reducing a fan-in characteristic of some portions of said second logic design in accordance with a first fan-in value.

8. A method as defined in claim 1, further comprising the step of applying a cell reduction sequence of logic transformations for reducing the number of cells in said second logic design, said cell reduction sequence being applied between said depth reduction and size reduction sequences.

9. A method as defined in claim 8, wherein said cell reduction sequence of logic transformations includes a first set of logic transformations followed by a second set of logic transformations, said second set of logic transformations comprising said depth reduction sequence of logic transformations.

10. A method as defined in claim 8, wherein said desired technology has a maximum allowable fan-in value, said step of applying said size reduction sequence of logic transformations comprising applying a first logic transformation (e.g., FACTORN) for reducing a fan-in characteristic of some portions of said second logic design in accordance with a desired fan-in value less than said maximum allowable fan-in value.

11. A method as defined in claim 10, further comprising the step of correcting the fan-in characteristics of said second logic design in accordance with said maximum allowable fan-in value subsequent to application of said size reduction sequence of logic transformations.

12. A method as defined in claim 1, wherein said depth reduction sequence of logic transformations is applied a plurality of times prior to applying said size reduction sequence of logic transformations.

13. A method as defined in claim 1, wherein said first logic design is implemented in AND/OR logic and said second logic design is implemented in NAND logic.

14. A method as defined in claim 1, wherein said first logic design is implemented in AND/OR logic and said second logic design is implemented in NOR logic.

15. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating, via said computer, a first logic circuit design in a first logic system in accordance with said description;

simplifying said first logic design via said computer;

converting, via said computer, said first logic design to a second logic design in a second logic system requiring fewer different logic operators than in said first logic design, said second logic design comprising a plurality of interconnected cells and performing equivalent functions as said first logic design;

simplifying said second logic design via said computer;

converting, via said computer, said simplified second logic design to a hardware design in said desired technology comprising a plurality of interconnected hardware components; and

simplifying said hardware design via said computer, said step of simplifying said hardware design comprising: applying a first hardware transformation

set for substituting technology specific components for predetermined patterns of said hardware;

dotting signal lines via said computer, to decrease the number of components in said hardware logic design and to decrease fan-in in some portions of said hardware logic design even if the number of components is said portions is not decreased;

adjusting path lengths in said hardware logic design via said computer; and

correcting fan-out in said hardware logic design to a desired value via said computer.

16. A method as defined in claim 15, wherein said step of simplifying said hardware design further comprises applying said first hardware transformation set again after said dotting step but before said adjusting step.

17. A method as defined in claim 16, wherein said step of simplifying said hardware logic design further comprises the step of correcting said fan-out in said hardware logic design after said second application of said first hardware transformation set and before said adjusting step

18. A method as defined in claim 15, wherein said hardware logic design includes inverters receiving and inverting outputs from associated components, and wherein, when said desired technology is a dual-rail technology, said step of simplifying said hardware design further comprises the step of applying a dual-rail transformation for removing some of said inverters by substituting for said inverter and opposite-phase output signal available from its associated component, said dual-rail conversion transformation being applied both prior to said step of applying said first hardware transformation and subsequent to said step of correcting fan-out.

19. A method as defined in claim 18, wherein said dual-rail conversion transformation is applied prior to said step of applying said first hardware transformation without regard to the effect of said dual-rail conversion transformation on fan-out characteristics of said hardware logic design, said dual-rail conversion transformation being applied after said step of correcting fan-out only to the extent that application of said dualrail conversion transformation will not result in fan-out exceeding said desired value.

20. A method as defined in claim 15, wherein said hardware logic design includes inverters receiving and inverting outputs from associated components, and wherein, when said desired technology is a dual-rail technology, said step of simplifying said hardware design further comprises the step of applying a dual-rail comversion transformation, prior to said step of applying said first hardware transformation, for removing some of said inverters by substituting for said some inverters an oppositephase output available from their associated components.

21. A method as defined in claim 20, wherein said step of simplifying said hardware design further comprises the step of applying said dual-rail conversion transformation subsequent to said step of correcting fan-out.

22. A method as defined in claim 21, wherein said dual-rail conversion transformation is applied prior to said application of said first hardware transformation without regard to the effect of said dual-rail conversion transformation on fan-out characteristics of said hardware logic design, and is applied subsequent to said step of correcting fan-out only to the extent that application of said dual-rail conversion transformation will not result in fan-out exceeding said desired value.

23. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating a first logic circuit design via said computer, in accordance with said description;

simplifying said first logic design via said computer;

converting, via said computer, said first logic design to a second logic design in a second logic system requiring fewer different logic operators than in said first logic design;

simplifying said second logic design via said computer;

converting, via said computer, said simplified second logic design to a hardware design in said desired technology comprising a plurality of interconnected hardware components and including inverters for receiving and inverting output signals from associated ones of said components; and

simplifying said hardware design via said computer, said step of simplifying said hardware logic design comprising: applying a dual-rail conversion transformation for removing some of said inverters by substituting for said some inverters an opposite phase output signal available from their associated components, said dual-rail conversion transformation being applied without regard to the effect on fan-out characteristics of said hardware logic design; applying a first hardware transformation set for substituting technology-specific components; dotting signal lines in said hardware logic design; adjusting path lengths in said hardware logic design; correcting fan-out in said hardware logic design to a desired value; and applying said dual-rail conversion transformation only to the extent that it does not result in a fan-out exceeding said desired value.

24. An automated logic synthesis method of designing, on a computer, a logic circuitry implementation in a desired technology from input data to said computer comprising a description of operating characteristics to be provided by said logic circuitry, said method comprising the steps of:

generating, via said computer, a first logic circuit design accordance with said description;

simplifying said first logic design via said computer;

converting, via said computer, said first logic design to a second logic design in a second logic system requiring fewer different logic operators than in said first logic design;

simplifying said second logic design via said computer;

converting, via said computer, said simplified second logic design to a hardware design in said desired technology comprising a plurality of interconnected hardware components; and

simplifying said hardware design via said computer, said step of simplifying said hardware logic design comprising selectively applying first first or second hardware transformation sets for substituting technology-specific components for predetermined patterns of said hardware components, said first hardware transformation set resulting in fewer components than said second hardware transformation set and said second hardware transformation set resulting in higher-speed logic than said first hardware transformation set.

* * * * *

5

US005197016A

United States Patent [19]

Sugimoto et al.

[11] Patent Number: 5,197,016

[45] Date of Patent: Mar. 23, 1993

[54] **INTEGRATED SILICON-SOFTWARE COMPILER**

[75] Inventors: **Tai Sugimoto; Hideaki Kobayashi,** both of Columbia, S.C.; **Masahiro Shindo; Haruo Nakayama,** both of Osaka, Japan

[73] Assignees: **International Chip Corporation,** Columbia, S.C.; **Ricoh Company, Ltd.,** Tokyo, Japan

[21] Appl. No.: **380,079**

[22] Filed: **Jul. 14, 1989**

Related U.S. Application Data

[63] Continuation-in-part of Ser. No. 143,821, Jan. 13, 1988, Pat. No. 4,922,432.

[51] **Int. Cl.**[5] **G06F 15/60**

[52] **U.S. Cl.** **364/490;** 364/489; 364/488

[58] **Field of Search** 364/488, 489, 490, 491

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,648,044	3/1987	Hardy et al.	395/76
4,658,370	4/1987	Erman et al.	395/76
4,675,829	6/1987	Clemenson	395/65
4,703,435	10/1987	Darringer et al.	364/489
4,922,432	5/1990	Kobayashi et al.	364/490

OTHER PUBLICATIONS

"A Front End Graphic Interface to the First Silicon Compiler" by J. H. Nash et al., European Conf. on Electronic Design Automation (EDA84), pp. 120–124.

An Engineering Approach to Digital Design, William I. Fletcher, Prentice-Hall, Inc. pp. 491–505.

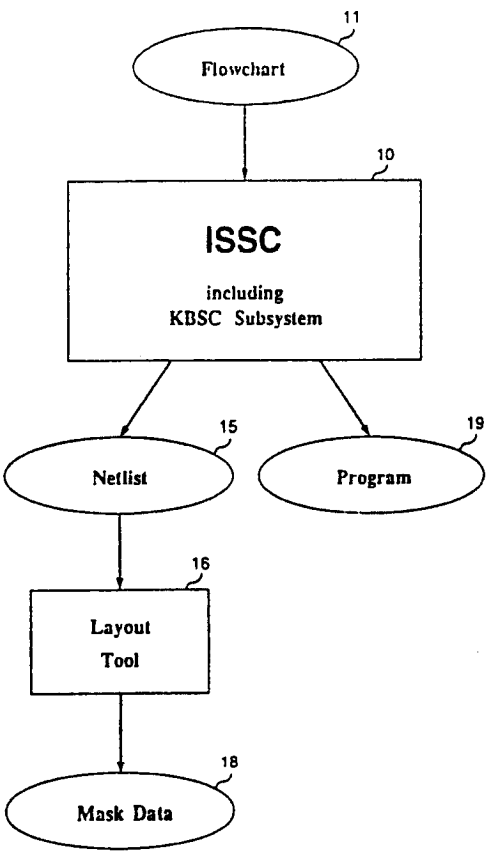
Primary Examiner—Vincent N. Trans

Attorney, Agent, or Firm—Bell, Seltzer, Park & Gibson

[57] **ABSTRACT**

A computer-aided system and method is disclosed for designing an application specific integrated circuit (ASIC) whose intended function is implemented both by a hardware subsystem including hardware elements on the integrated circuit and by a software subsystem including a general purpose microprocessor also on the integrated circuit. The system also generates software instructions for use by the software subsystem. The system utilizes a knowledge based expert system, with a knowledge base extracted from expert ASIC designers, and thus makes it possible for ASIC's to be designed and provided quickly and economically by persons not having the highly specialized skill of an ASIC designer.

32 Claims, 7 Drawing Sheets



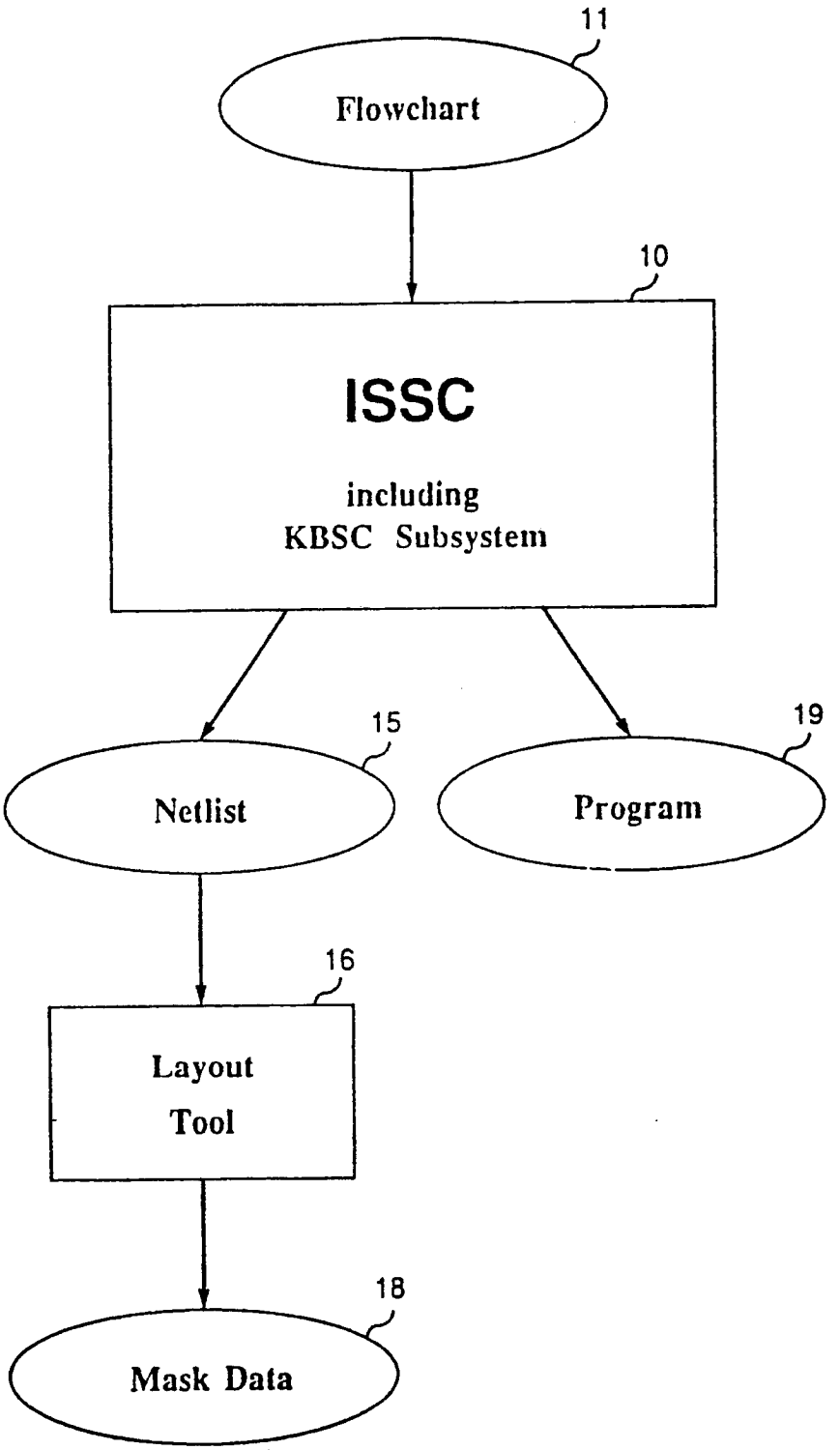

11
Flowchart
10
ISSC
including
KBSC Subsystem
15
Netlist
19
Program
16
Layout
Tool
18
Mask Data

FIG. 1.

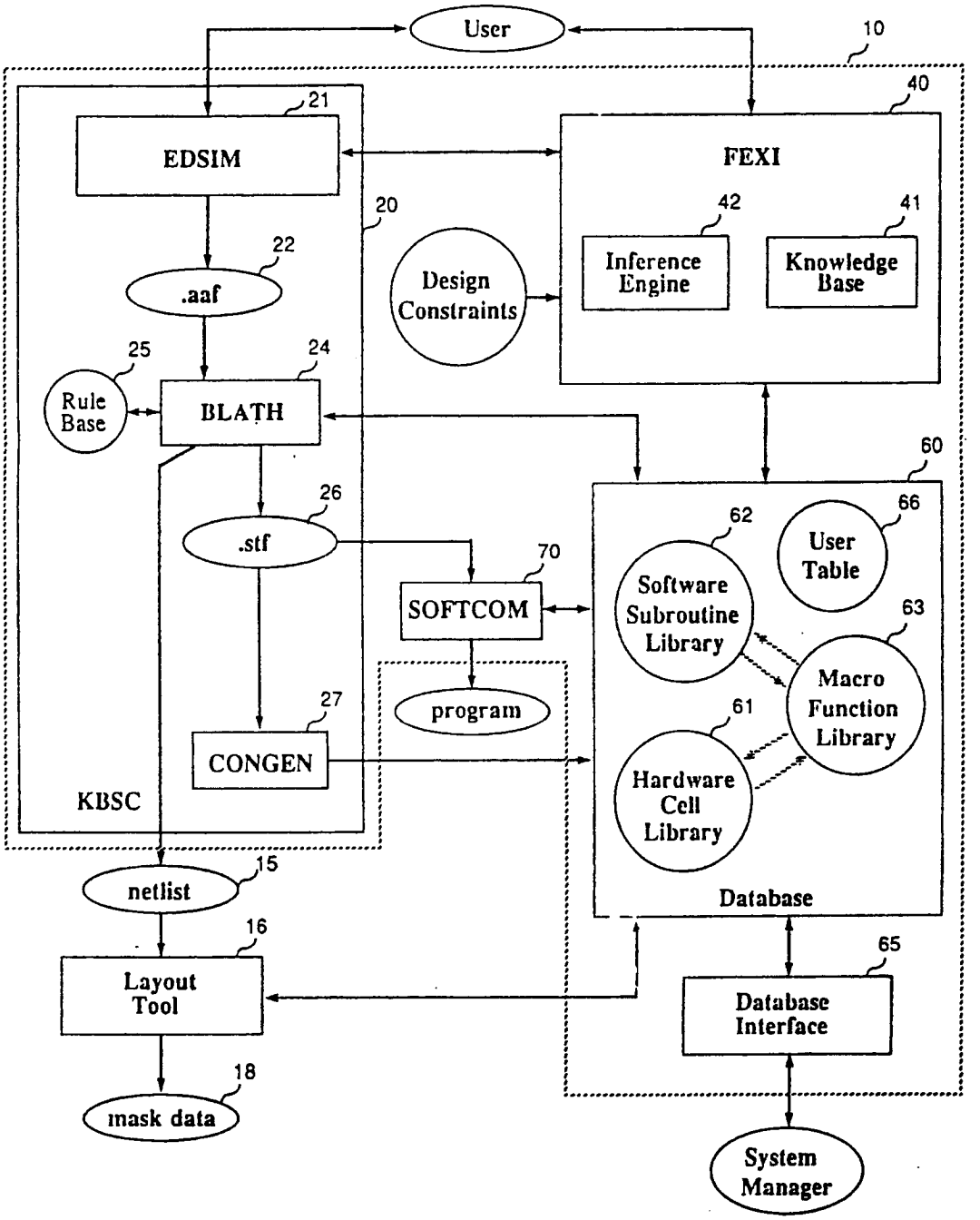

User
10
21
EDSIM
40
FEXI
20
42
41
Inference Engine
Knowledge Base
Design Constraints
22
.aaf
25
24
Rule Base
BLATH
60
26
.stf
62
66
User Table
70
SOFTCOM
Software Subroutine Library
63
Macro Function Library
program
27
61
CONGEN
Hardware Cell Library
KBSC
15
netlist
Database
16
65
Layout Tool
Database Interface
18
mask data
System Manager

FIG. 2.

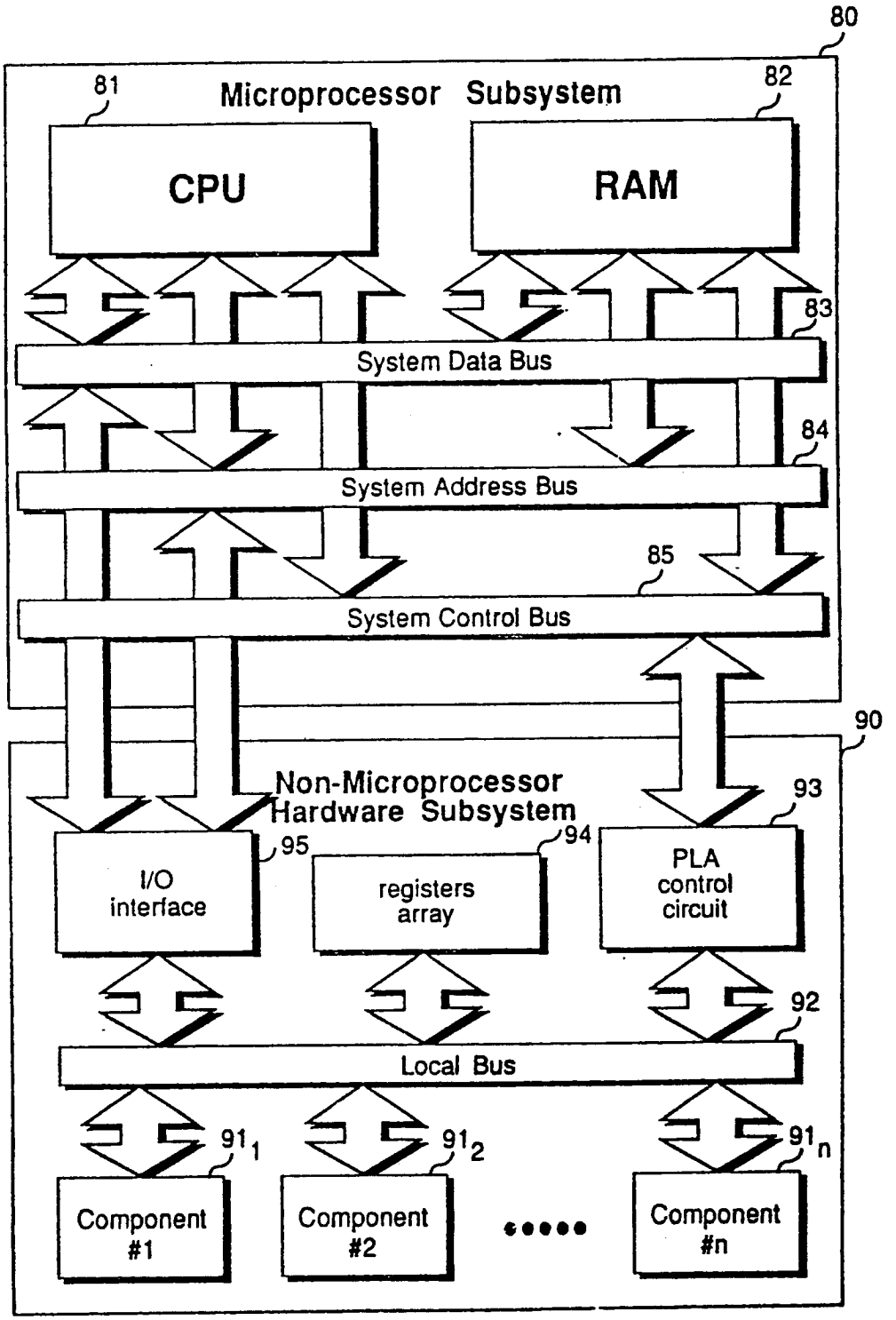


FIG. 3.

DEF017268

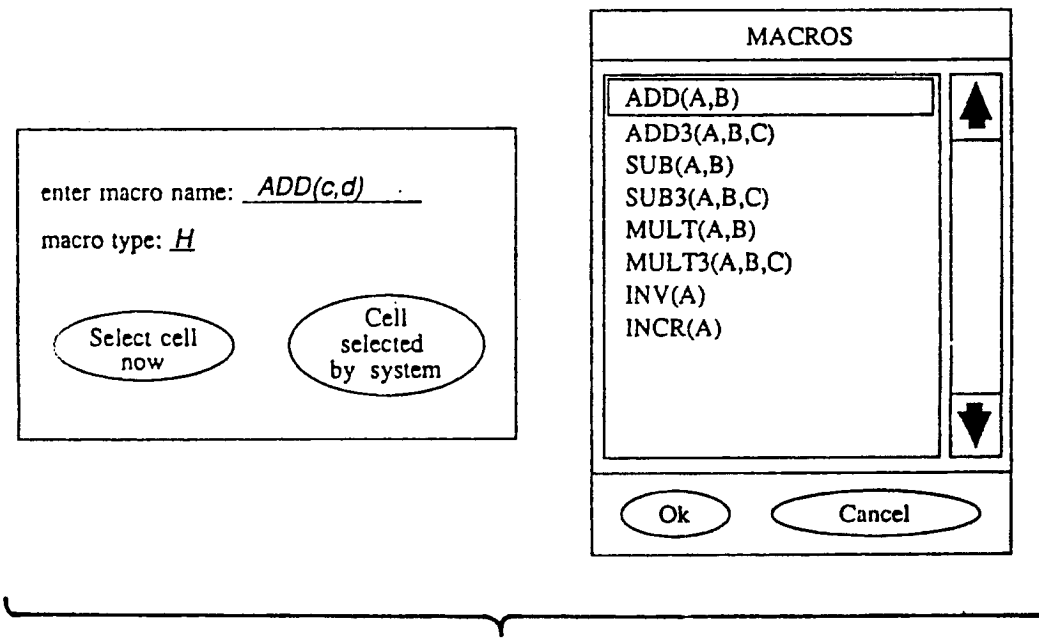

enter macro name: ADD(c,d)
macro type: H
Select cell now
Cell selected by system
MACROS
ADD(A,B)
ADD3(A,B,C)
SUB(A,B)
SUB3(A,B,C)
MULT(A,B)
MULT3(A,B,C)
INV(A)
INCR(A)
Ok
Cancel

FIG. 4.

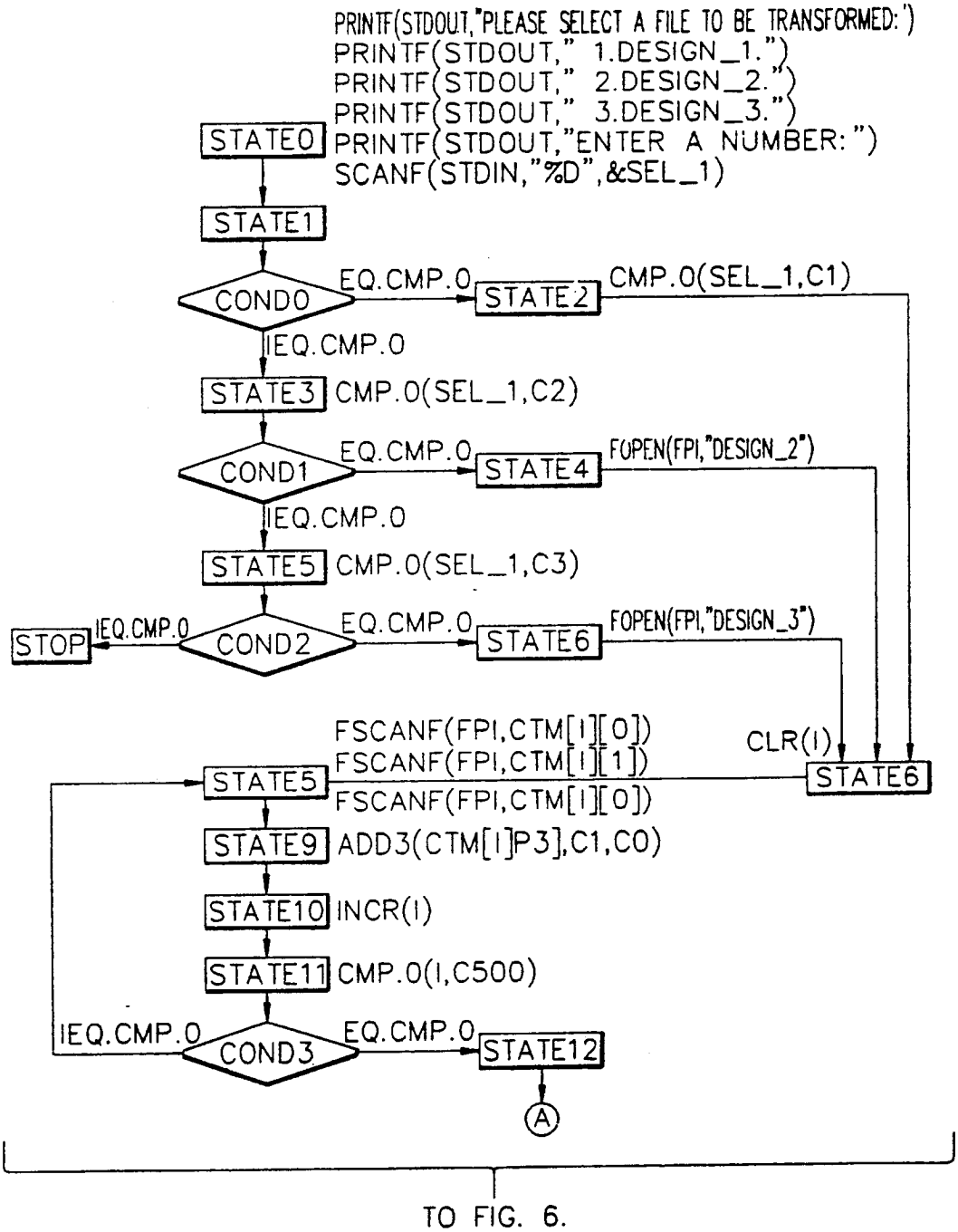


FIG. 5.

DEF017270

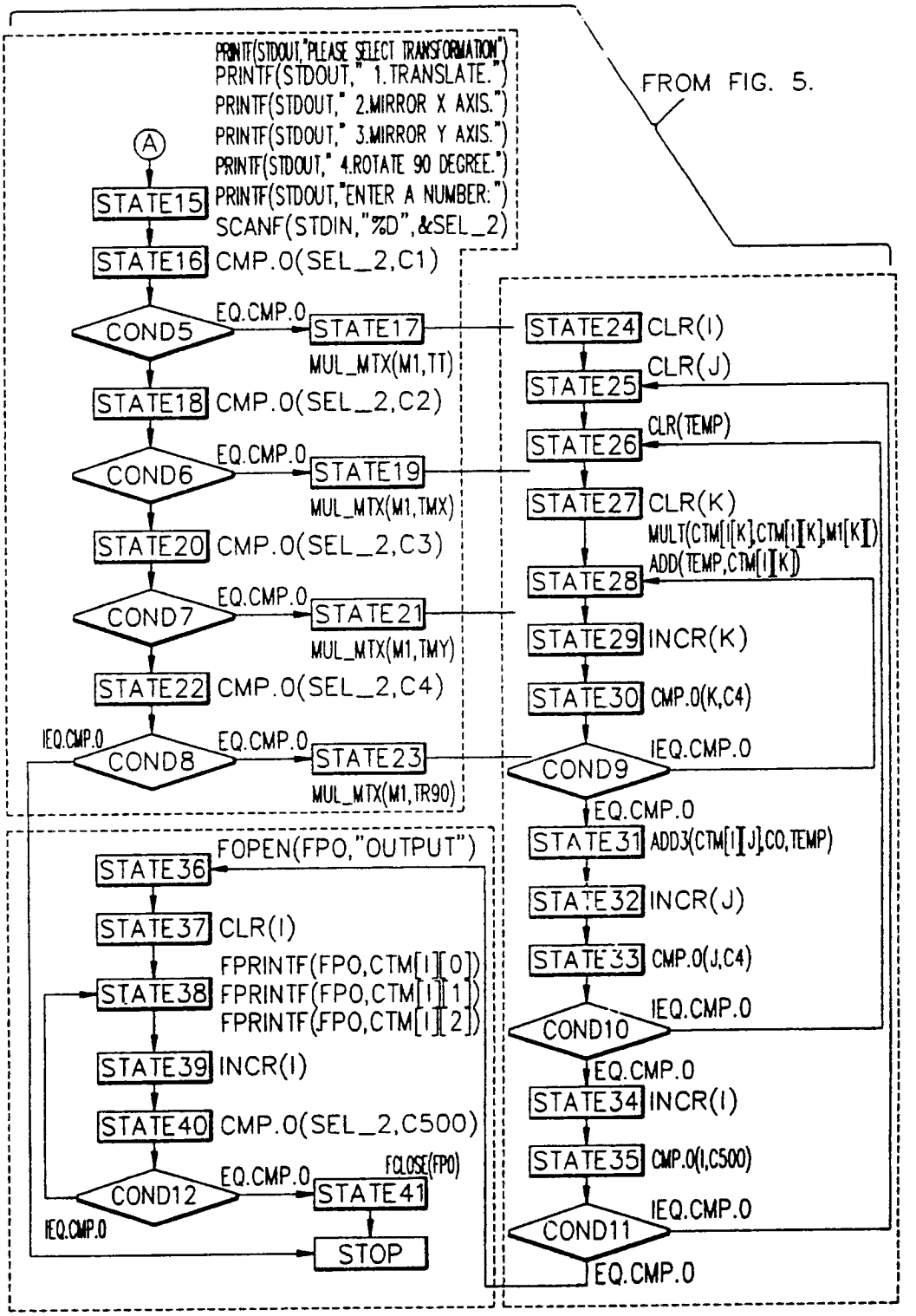


FIG. 6.

DEF017271

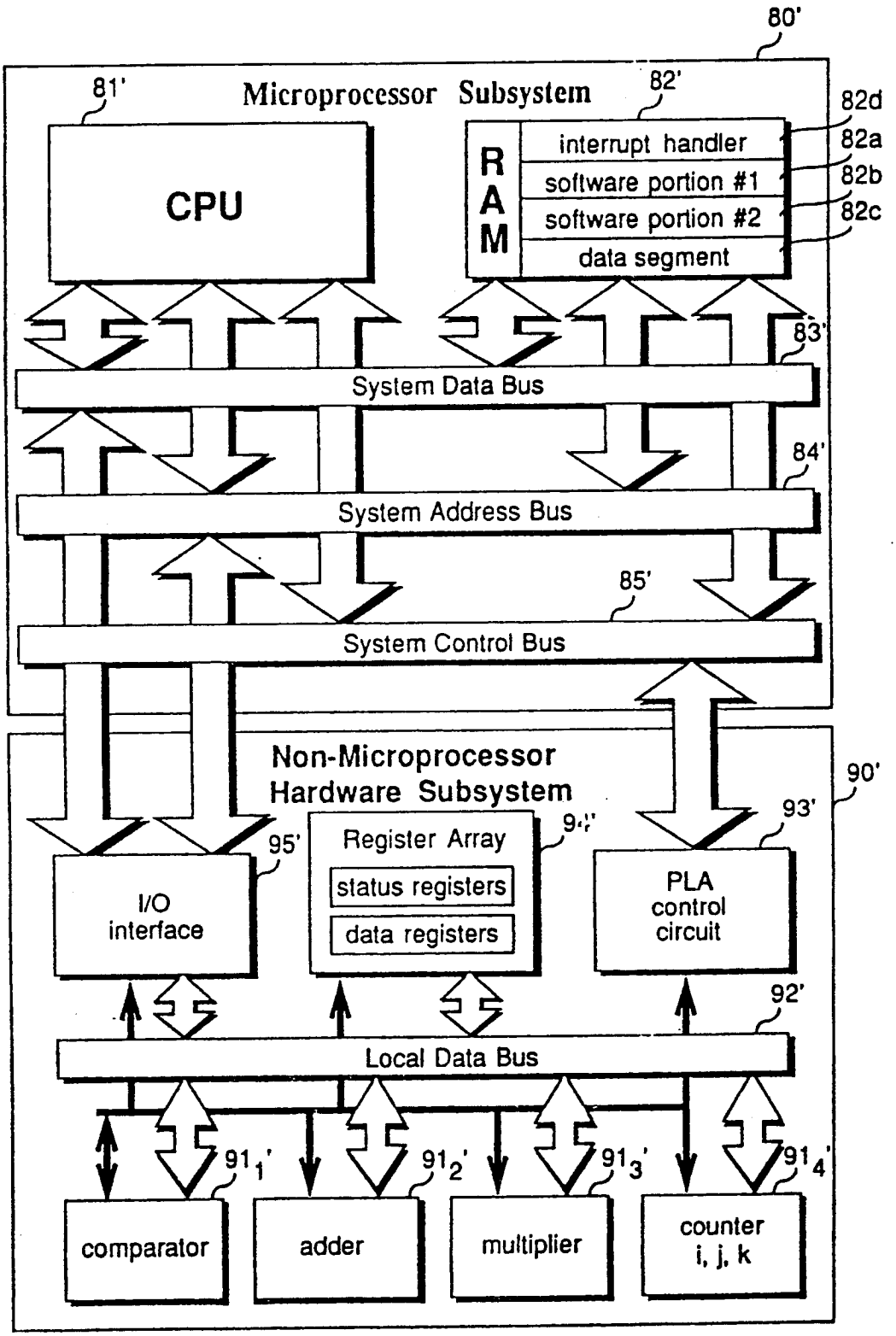


FIG. 7.

DEF017272

INTEGRATED SILICON-SOFTWARE COMPILER

CROSS-REFERENCE TO RELATED APPLICATION

This application is a continuation-in-part of copending U.S. application Ser. No. 143,821, filed Jan. 13, 1988, now U.S. Pat. No. 4,922,432.

FIELD AND BACKGROUND OF THE INVENTION

This invention relates to the design of integrated circuits, and more particularly relates to a computer-aided system and method for designing application specific integrated circuits.

An application specific integrated circuit (ASIC) is an integrated circuit chip designed to perform a specific function, as distinguished from standard, general purpose integrated circuit chips, such as microprocessors, memory chips, etc. A highly skilled design engineer having specialized knowledge in VLSI circuit design is ordinarily required to design an ASIC. In the design process, the VLSI design engineer will consider the particular objectives to be accomplished and tasks to be performed by the integrated circuit and will create structural level design specifications which define the various hardware components required to perform the desired function, as well as the interconnection requirements between these components. A system controller must also be designed for synchronizing the operations of these components. This requires an extensive and all encompassing knowledge of the various hardware components required to achieve the desired objectives, as well as their interconnection requirements, signal level compatibility, timing compatibility, physical layout, etc. At each design step, the designer must do tedious analysis. The design specifications created by the VLSI design engineer may, for example, be in the form of circuit schematics, parameters or specialized hardware description languages (HDLs).

From the structural level design specifications, the description of the hardware components and interconnections is converted to a physical chip layout level description which describes the actual topological characteristics of the integrated circuit chip. This physical chip layout level description provides the mask data needed for fabricating the chip.

Due to the tremendous advances in very large scale integration (VLSI) technology, highly complex circuit systems are being built on a single chip. With their complexity and the demand to design custom chips at a faster rate, in large quantities, and for an ever increasing number of specific applications, computer-aided design (CAD) techniques need to be used. CAD techniques have been used with success in design and verification of integrated circuits, at both the structural level and at the physical layout level. For example, CAD systems have been developed for assisting in converting VLSI structural level descriptions of integrated circuits into the physical layout level topological mask data required for actually producing the chip. Although the presently available computer-aided design systems greatly facilitate the design process, the current practice still requires highly skilled VLSI design engineers to create the necessary structural level hardware descriptions.

Only a small number of VLSI designers possess the highly specialized skills needed to create structural level integrated circuit hardware descriptions. Even

with the assistance of available VLSI CAD tools, the design process is time consuming and the probability of error is also high because of human involvements. There is a very significant need for a better and more cost effective way to design custom integrated circuits.

In U.S. application Ser. No. 143,821, filed Jan. 13, 1988, and entitled *Knowledge Based Method and Apparatus for Designing Integrated Circuits Using Functional Specifications*, there is disclosed a computer-aided design system and method which enables a user to define the functional requirements for a desired application specific integrated circuit using an easily understood architecture independent functional level representation, such as a flowchart. From this functional level description, a computer implemented expert system generates the detailed structural level definitions needed for producing the application specific integrated circuit. The structural level definitions include a list of the integrated circuit hardware cells needed to achieve the functional specifications. Also included in the detailed structural definitions are the data paths among the selected hardware cells, a system controller for coordinating the operation of the cells and control paths for the selected integrated circuit cells. The various hardware cells are selected from a cell library of previously designed hardware cells of various functions and technical specifications. From this detailed structural level definition it is possible, using either known manual techniques or existing VLSI CAD layout systems to generate the detailed chip level geometrical information (e.g. mask data) required to produce the particular application specific integrated circuit in chip form.

The system described in the aforementioned copending application provides a very significant advance over the methods previously available for designing application specific integrated circuits and opens the possibility for the design and production of application specific integrated circuits by designers, engineers and technicians who may not possess the specialized expert knowledge of a highly skilled VLSI design engineer.

SUMMARY OF THE INVENTION

The present invention provides an improvement over and an extension to the system and method of the aforementioned copending application. Although the system and method of the aforementioned application provides an excellent means for designing ASICs whose functions are implemented in hardware form, there are some occasions where particular functions of the ASCI would best be implemented by software.

The present invention provides a computer-aided system and method for designing an application specific integrated circuit whose intended function is implemented by both a hardware subsystem including hardware elements on the integrated circuit and by a software subsystem including a general purpose microprocessor. Thus, the resulting ASIC produced by the computer-aided design system and method of the present invention includes, on a single integrated circuit chip, a microprocessor for executing the software instructions of the software subsystem and various integrated circuit hardware cells for performing other functions of the integrated circuit.

The system more particularly utilizes a knowledge based expert system, with a knowledge base extracted from expert ASIC designers with a high level of expertise in system design. The knowledge base contains

rules for selecting software subroutines from a software subroutine library of predefined functions or operations, as well as rules for selecting hardware cells from a cell library of predefined cells of various types and functions. An inference engine is provided for selecting appropriate hardware cells or software subroutines from these libraries in accordance with the rules of the knowledge base.

The functional specifications of the desired ASIC are independent of any particular architecture or design style and can be defined in a suitable manner, such as in text form or preferably in a flowchart format. The flowchart is a highly effective means of describing a sequence of logical operations, and is well understood by software and hardware designers of varying levels of expertise and training. The designer, when defining the series of operations which implement the intended function of the application specific integrated circuit, may specify whether a particular operation should be implemented in hardware or in software.

The designer may also specify design constraints for the ASIC, such as annual volume, speed, size, pin count, packaging type, power consumption and thermal stability. The knowledge base contains design style rules for selecting, based upon the specified design constraints, an optimum design style for implementation of the hardware subsystem employing various technologies such as programmable logic device (PLD), gate array, standard cell and macro cell.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be better understood by reference to the detailed description which follows, taken in connection with the accompanying drawings, in which

FIG. **1** is a block schematic diagram showing how integrated circuit mask data is created from flowchart descriptions by the system of the present invention;

FIG. **2** is a block schematic diagram showing the various elements of the integrated silicon-software compiler system and method of the present invention;

FIG. **3** is a schematic illustration of the system configuration of an ASIC containing both a microprocessor subsystem and a non-microprocessor hardware subsystem;

FIG. **4** is an illustration of a computer display screen which the designer uses in defining the functional specifications of the integrated circuit;

FIGS. **5** and **6** are flowcharts illustrating the functions to be performed by an ASIC in an illustrative design example of the system and method of the invention; and

FIG. **7** is a schematic illustration similar to FIG. **3**, but showing the particular ASIC of the design example.

MORE DETAILED DESCRIPTION OF THE INVENTION

System Overview

The overall system flow in accordance with the present invention is illustrated in FIG. **1**. The computer-aided system and method of the present invention is represented by the block **10**, and for simplicity is referred to herein by the acronym ISSC (integrated silicon-software compiler). The ISSC **10** receives as its input (which are technology and architecture independent) specifications **11**, and which define on a functional or behavioral level the functions which are to be performed by the target application specific integrated circuit (ASIC). As illustrated in FIG. **1**, the user enters the functional specifications of the desired target ASIC into the integrated silicon-software compiler (ISSC) **10** in the form of a flowchart **11**. The ISSC **10** then generates a netlist **15** from the flowchart. The netlist **15** includes architecture specific definitions of the hardware cells required to implement the functions of the hardware subsystem, a definition of the microprocessor used to implement the functions of the software subsystem, and definitions of a system controller, data paths and control paths for interconnecting the various components. The netlist **15** can be used as input to any existing VLSI layout and routing tool **16** to create mask data **18** for geometrical layout. The ISSC system **10** also generates a program **19** from the flowchart information for implementing the functions of the software subsystem. The program **19** can be stored in an external or internal memory associated with the microprocessor on the target ASIC.

The ISSC system **10** can be operated on a suitable programmed general purpose digital computer. By way of example, one embodiment of the system is operated in a workstation environment such as Sun3 and VAX-Station-II/GPX running UNIX Operating System and X Window Manager. The software uses C programming language and a data base such as INGRES or Gbase. The human interface is mainly done by the use of a pointing device, such as a mouse, with pop up menus, buttons, and a special purpose command language. The permanent data of the integrated circuit design are stored in a database for easy retrieval and update. Main memory temporarily stores the executable code, design data (flowchart, logic, etc.), data base (cell library), and knowledge base. The CPU performs the main tasks of creating and simulating flowcharts and the automatic synthesis of the design.

The primary elements or modules which comprise the ISSC system are shown in FIG. **2**. In the embodiment illustrated and described herein, these elements or modules are in the form of software programs, although persons skilled in the appropriate art will recognize that these elements can be easily embodied in other forms, such as in hardware for example.

Referring more particularly to FIG. **2**, it will be seen that the ISSC system **10** includes a module or subsystem **20** called KBSC (Knowledge Based Silicon Compiler). The KBSC subsystem **20** corresponds to the system and method described in copending U.S. application Ser. No. 143,821, filed Jan. 13, 1988, the subject matter of which is incorporated herein by reference. As shown in FIG. **2**, the KBSC subsystem **20** includes a number of modules or programs which collectively provide an interface with the user for receiving input of the functional specifications for the particular target ASIC and which provide as output a netlist **15** and a program **19**. Other major subsystems or modules of the ISSC **10** include a front-end expert system interface (FEXI) **40**, a database **60** and a software compiler (SOFTCOM) **70**.

The KBSC subsystem **20** includes a program **21** called EDSIM (EDitor SIMulator), which comprises a flowchart editor for creating and editing flowcharts and a flowchart simulator for simulation and verification of flowcharts. The output of EDSIM **21** is an intermediate file **22**, referred to as an Antecedent-Action-Form (AAF), which contains a behavioral description of the system that is to be designed. The AAF file **22** contains information about storage elements which compose the data paths, memory elements and external connections

to the system. The AAF file **22** is the input to the BLATH program **24**.

The BLATH (Block Level Aaf To Hardware) program **24** is a knowledge based logic synthesis program which selects optimum hardware cells for the hardware subsystem from the hardware cell library and also selects a microprocessor or CPU megacell for use by the software subsystem. The selection is based upon functional descriptions in the flowchart, as specified by the macros assigned to each action represented in the flowchart. BLATH **24** uses a knowledge base **25** extracted from VLSI design experts to select the hardware cells and microprocessor based upon design constraint rules, design style selection rules, and software/hardware routine selection rules. BLATH **24** selects from a database **60** which includes a hardware cell library **61** of previously designed hardware cells and a software subroutine library **62** of previously designed software subroutines. BLATH **24** also outputs an STF file **26** which is used by a system controller generator CONGEN (CONtroller GENerator) **27** and by the software compiler (SOFTCOM) **70**. The controller generator **27** generates a custom designed system controller for controlling the operation of the hardware cells and coordinating with the microprocessor of the software subsystem. Thus, with a functional flowchart input from EDSIM **21**, BLATH **24** selects a microprocessor and all required hardware cells, generates data and control paths, and generates a netlist **15** describing all of ASIC design information.

Target ASIC Design

FIG. **3** illustrates the system configuration of an application specific integrated circuit (ASIC) designed using the ISSC **10**. It consists of two subsystems, a microprocessor subsystem **80** and a non-microprocessor hardware subsystem **90**. The microprocessor subsystem contains a general purpose microprocessor CPU **81** and random access memory (RAM) **82** to execute the software functions of the ASIC. The software program is stored in RAM memory **82**. The hardware subsystem **90** contains special purpose hardware components or cells which perform the functions of the ASIC. Functions are executed in special hardware components primarily to reduce processing time. The various hardware components 91_1, 91_2 . . . 91_n are integrated into a subsystem which includes a local bus **92**, a PLA control circuit **93**, a register array **94** and an input/output interface **95**. These two subsystems communicate with each other through a system data bus **83**, a system address bus **84** and a system control bus **85**.

EDSIM

The creation and verification of the flowchart is the first step in the VLSI design methodology. The translation from an algorithm to an equivalent flowchart is performed with the flowchart editor, which is contained within the program EDSIM **21**. The flowchart editor provides a working environment for interactive flowchart editing with a designer friendly interface. A graphical display of the flowchart is provided consisting of boxes, diamonds, and lines. All are drawn on the screen and look like a traditional flowchart. The flowchart editor also provides functions such as loading and saving flowcharts. EDSIM will generate an intermediate file **22** for each flowchart. This file is then used by the BLATH program **24** to generate a netlist **15**.

The main editing functions of the flowchart editor include, create, edit, and delete states, conditions, and transitions. The create operation allows the designer to add a new state, condition, or transition to a flowchart. Edit allows the designer to change the position of a state, condition or transition, and delete allows the designer to remove a state, condition or transition from the current flowchart. States which contain actions are represented by boxes, conditions are represented by diamonds, and transitions are represented by lines with arrows showing the direction of the transition.

Once the states and transitions have been created, the designer assigns operations to each state. These operations are made up of macro functions and arguments. The macro function library **63** in database **60** contains a set of macros defining various operations corresponding to the available actions and conditions which can be specified in a flowchart. During the operation of the EDSIM program **21**, the user assigns to each block in the flowchart a macro selected from the macro library and the associated parameters of the macro, if any. FIG. **4** illustrates a screen provided for the designer to specify a macro function in a state in the flowchart. The designer types a macro name along with its parameters in the field associated with the label "enter macro name". In the righthand portion of the screen a macro list is displayed for the designer to select a macro, which can be selected by pointing and clicking a mouse. Once a macro has been selected by this method, the designer would then specify the parameters of the macro. The designer can also select the desired macro type, e.g. hardware (H) or software (S). He then clicks on one of the two circles, "select cell now" or "cell selected by system". If "select cell now" is selected, then EDSIM **21** creates and sends a query to FEXI **40**. The expert system FEXI then queries the database and retrieves suitable cells (either hardware or software as desired by the user). If neither is selected by the user, then both types of cells are displayed. FEXI displays this cell list to the designer and waits for him to select a cell. The designer can query about the features of the cells. EDSIM then returns to normal flowchart editing. If the designer had selected "cell selected by system" from the above screen, then FEXI is not invoked and normal flowchart editing continues.

A list of basic macro functions available in the macro function library **63** is shown in Table 1.

TABLE 1

id	macro name	parameters	description
1	ADD	2	B = A + B
2	ADD3	3	C = A + B
3	SUB	2	B = A − B
4	MULT	3	C = A * B
5	DIV	3	C = A/B
6	DEC	1	A = A − 1
7	INCR	1	A = A + 1
8	REG	2	B = A
9	CMP	2	compare A and B and set EQ, LT or GT signals
10	CMP0	1	compare A with 0 and set EQ, LT or GT signals
11	NEGATE	1	A = NOT(A)
12	MOD	3	C = A Modulus B
13	POW	3	C = A B
14	DC2	5	decode A into B, C, D, and E
15	EC2	5	encode A, B, C, and D into E
16	MOVE	2	B = A

FEXI

The front-end expert system interface FEXI **40** is invoked by EDSIM **21** for obtaining user design constraints. FEXI starts an interactive session with the user to determine design constraints for the target ASIC such as the following: speed, die size, number of I/O pins, development time, chip count, desired yield, chip cost, package type, technology, power consumption, production volume, pin count. design style, microprocessor type, etc. Once the user has answered the necessary inquiries regarding the design constraints, FEXI **40** utilizes a knowledge base **41** and an inference engine **42** to determine an optimum design style for the hardware components of the integrated circuit from such choices as PLD (programmable logic device), gate array, standard cell, and macro cell. It then determines the technology to be used, and finally selects a hardware cell library to be used. For the software subsystem, a microprocessor is selected among various established microprocessor designs, such as Intel 8086, Motorola 68000, Intel 80286, etc. The database **60** includes software subroutine libraries for each of the standard microprocessors. Once the microprocessor type has been selected, the appropriate software subroutine library for that microprocessor type is selected. FEXI operates as a separate program which runs in parallel with EDSIM **21**. Although illustrated in FIG. **2** separately, FEXI uses the same inference engine used by BLATH **24**.

AAF

The AAF file **22** (Antecedent-Action-Form) is the input to BLATH **24**. It contains the behavioral description of the system that is to be designed. AAF file **22** contains information about storage elements which compose the data paths, memory elements and external connections to the system. This information is contained in the first section of the file. The rest of the file contains a description of the behavior of the system. The first line of the AAF file **22** contains the name of the system and is described by the following form:

name <name>

where <name> is the name of the system.

Storage elements are described at the top of the AAF file **22**. Storage elements can be data paths, external connections or memory elements. They are defined using the "data path", "data mask", "rom" and "ram" statements. These statements are described by the following forms:

"data path" <list>
"data mask" <list>
"rom" <list>
"ram" <list>

where <list> is a list of definitions. These lists are further described below.

Data Paths are defined in the "data path" statement. Data paths are described by the following form:

<pathname>'<'<startbit>':'<stopbit>'>'

<Pathname> is the name of the data path. <Startbit> and <stopbit> are the beginning and ending bit positions of the path. Data Paths must be defined before they can be used by the rest of the AAF.

External Connections behave just like data paths except that they define an external connection to the system. An external connection is described just like a data path in the "data path" statement, except that it is preceded by an "⊥" to indicate that it is an external connection.

Data mask provide a means of specifying a partial bitfield that is to be treated as another data path. Data mask are described by the following form:

<maskname>'<'<startbit>':'<stopbit>'>'"='
<pathname>'<'<startbit>':'<stopbit>'>'

<Maskname> is the name of the mask that is to be treated as a data path. <Pathname> is the name of the data path that the partial bitfield comes from. <Startbit> and <stopbit> describe the bitrange of the mask and the bitfield of the data path.

Memory elements define RAMs and ROMs to be used in the system. Memories are defined in the "rom" and "ram" statements. Memory elements are described by the following form:

<memname> '[' <startaddr> ':' <stopaddr> ']' <startbit> ':' <stopbit>

where <memname> is the name of the memory. If the memory is a rom, then this name is also the name of a file containing the contents of the rom. <Startaddr> and <stopaddr> are the beginning and ending addresses of the memory. <Startbit> and <stopbit> are the beginning and ending bit positions of the word contained in the memory.

An example is given to illustrate the top part of an AAF file.

name	cpuexample;
data path	acc<0:15>, ir<0:15>, mar<0:15>, iar<0:15>, mdata<0:15>, sum<0:15>;
data mask	adr<0:9> = ir<0:9>;
rom	mem[0:65535]<0:7>;

The behavior of the system is described by state transition information and actions. The state transition information describes the control flow of the system. The actions describe what happens at each state. The representation of the state transition and action information is described in the following sub systems.

State transitions are defined as either being direct or conditional. Direct transitions are described by the following form:

<state1>:<state2>

where <state1> and <state2> are two state names. Direct transitions are an unconditional transition from state1 to state2. This transition will always occur when the system is in state1.

Conditional transitions are described by the following form:

<state1>:.<condition> <state2>

where <state1> and <state2> are the names of two states. <Condition> is the condition that must evaluate to true in order for this transition to be made. The condition is described as an AND function where signals and their compliments are ANDed. Compliments are shown by a "!" and ANDing is described by "*". Some sample conditions are shown below.

a*!b
!GT*!EQ
dog*cat*!bird

State actions consist of macros and assertions. Macros are described by the following form:

<macroname>'.'<instance>'(' <paramlist> ')'

<Macroname> is the name of the macro such as ADD or MOVE. <Instance> is a unique number for each instance of a macro. This allows distinctions to be made

between the same macro at different states. <Paramlist> is a list of zero or more parameter names separated by commas. Macros describe the actions that are to be taken when the system is at a given state. There are four types of parameters that can be used in a macro: Bus, Control, Signal, and Memory.

Bus parameters refer to the data paths or data mask that the macro is applied to. The data path or data mask must be defined in the appropriate statement at the top of the file. Optionally, a bitfield may be specified in the parameter list that will be used instead of the default of using the entire bitrange of the data path.

Control parameters indicate control signals that are necessary for the macro's actions. Controls are defined in the database definition of a macro and used by the rules.

Signal parameters indicate outputs from the macro that go to the controller to indicate the results of the macro. Signals are also defined in the database definition of a macro. Memory parameters refer to a memory that has been previously defined.

An example of the second part of an AAF is given. This example together with the previous example forms a complete AAF.

```
{
start:          ads;
ads  :          ift;
ift  :          dec;
dec  :          excstart;
excend:         end;
end  :          start;
lda  :          excend;
sta  :          excend;
add  :          excend;
bra  :          excend;
brp  :          excend;
excstart        :.      z2 lda;
excstart        :.      !z2*z3 sta;
excstart        :.      !z2*!z3*z4 add;
excstart        :.      !z2*!z3*!z4*z5*!z0 bra;
excstart        :.      !z2*!z3*!z4*z5*z0 brp;
ads     ::      MOVE.1( iar, mar );
ads     ::      INCR.1( iar );
ift     ::      STORE.1( mem, mar, mdata );
ift     ::      MOVE.2( mdata, ir );
dec     ::      MOVE.3( adr, mar );
dec     ::      DECODE.1( ir<10:15> );
lda     ::      STORE.2( mem, mar, mdata );
lda     ::      MOVE.4( mdata, acc );
sta     ::      LOAD.1( mem, mar, acc );
add     ::      STORE.3( mem, mar, mdata );
add     ::      ADD3.1( mdata, acc, sum );
add     ::      MOVE.5( sum, acc );
bra     ::      MOVE.6( ir<0:9>, iar );
brp     ::      MOVE.7( ir<0:9>, iar );
}
```

BLATH

To design a VLSI system from a flowchart description of a user application, it is necessary to match the functions in a flowchart with hardware cells or software subroutines from the hardware cell library **61** or the software subroutine library **62** (FIG. 2). This mapping preferably utilizes artificial intelligence techniques since the selection process is complicated and is done on the basis of a number of design parameters and constraints. The concept used for selection is analogous to that used in software compilation. In software compilation a number of subroutines are linked from libraries. In the design of VLSI systems, a functional macro can be mapped to members in the hardware cell library or software subroutine library. BLATH uses a rule based expert system to select the appropriate hardware cell or software subroutine to perform each action. If the hardware cell library has a number of cells with different geometries for performing the operation specified by the macro, then an appropriate cell can be selected on the basis of factors such as cell function, process technology used, time delay, power consumption, etc.

The knowledge base of BLATH contains information (rules) for:

1) selection of macros
2) merging of two macros
3) mapping of macros to cells
4) merging two cells
5) error diagnostics

The above information is stored in the knowledge base as rules. The first step of cell list generation is the transformation of the flowchart description into a block list. The block list contains a list of the functional blocks to be used in the integrated circuit. The BLATH maps the blocks to cells selected from the cell library, selecting an optimum cell for a block through the use of rules in the knowledge base. The rules which are applied at this point accomplish the following:

Map arguments to data paths
Map actions to macros
Connect these blocks

The rules used by BLATH have the following format: rule name

```
        if
                ( condition 1 )
                ( condition 2 )
                  ..
                  ..
                ( condition n )
        then
                ( action 1 )
                ( action 2 )
                  ..
                  ..
                ( action m ).
```

Exemplary of the rules used by BLATH are the following:

```
Rule 1
        IF      no blocks exist
        THEN
                generate a system controller.
Rule 2
        IF      a state exists which has a macro AND
                this macro has not been mapped to a block
        THEN
                find a corresponding macro in the library
                and generate a block for this macro.
Rule 3
        IF      there is a transition between two states AND
                there are macros in these states using the same
                argument
        THEN
                make a connection from a register corresponding to the first macro to another register
                corresponding to the second macro.
Rule 4
        IF      a register has only a single connection from
                another register
        THEN
                combine these registers into a single register.
Rule 5
        IF      there are two comparators AND
                input data widths are of the same size AND
                one input of these is same AND
                the outputs of the comparators are used to
```

-continued

		perform the same operation.
	THEN	
		combine these comparators into a single comparator.
Rule 6		
	IF	there is a data without a register
	THEN	
		allocate a register for this data.
Rule 7		
	IF	all the blocks have been interconnected AND a block has a few terminals not connected
	THEN	
		remove the block and its terminals, or issue an error message.
Rule 8		
	IF	memory is to be used, but a block has not been created for it
	THEN	
		create a memory block with data, address, read and write data and control terminals.
Rule 9		
	IF	a register has a single connection to a counter
	THEN	
		combine the register and the counter; remove the register and its terminals.
Rule 10		
	IF	there are connections to a terminal of a block from many different blocks
	THEN	
		insert a multiplexor; remove the connections to the terminals and connect them to the input of the multiplexor; connect the output of the multiplexor to the input of the block.

Additional rules address the following points:
remove cell(s) that can be replaced by using the outputs of other cell(s)
reduce multiplexor trees
use fan-out from the cells, etc.

Database

The database **60** stores information relating to hardware cells, software subroutines, macro functions, user information, and the like. A database interface **65** is provided to allow the system manager to make additions, deletions and modifications to the database. A user table **66** maintains information for every valid user of the ISSC system, including the user identification name, number and password. In the hardware cell library **61** various types of data are stored for each cell, including:

1) functional level information: description of the cell at the register transfer level
2) logic level information: description and terms of flip flops and gates
3) circuit level information: description at the transistor level
4) layout level information: geometrical mask level specifications

A cell table within the hardware cell library **61** contains a record for each cell in the cell library. Every cell in the cell table can be used as instance cell of a larger cell. Once a cell becomes an instance of a larger cell, a data dependency is created. Once this occurs, the contents in the cell can not be altered until the data dependency is eliminated. This information is kept in the attribute "times_used". The attributes of a cell as kept in the cell table are summarized in Table 2:

TABLE 2

Key	Name	Type	Integrity	Description
1	cell_id	i4	>=0	cell identification number
	cell_name	c12	alpha-numeric	cell name assigned by designer
	user_id		i4>=0	user id of the owner of cell
	width		i4>0	width of cell in centi-microns
	height		i4>0	height of cell in centi-microns
	cif_name	c20	unix file name	name of the cif file
	technology_id	i4	>=0	id of associated technology
	test_id		i4>=0	id of the test tool(s) used
	macro_id	i4	>=0	id of macro function for cell
	delay		f4>0	max. prop. delay in nano second
	power		i4>0	power consumption in micro watt
	net_name	c20	unix file name	name of the netlist file
	protection	i4	>=0	flags for user, group, world prot.
	times-used	i4	>=0	counts cell reference quantity
	date	date		date created

An example of a cell is shown below:

cell_id	23
cell_name	XIN02
user_id	8946
width	68
height	2.2
cif_name	xin02.cif
technology_id	1
test_id	1
macro_id	4
delay	0.3
power	25
net_name	xin02.mdl
protection	555
date	88-11-10

Each terminal for a cell is kept in a terminal table. Terminals can be signal, power or ground. The cell_id determines which cell the terminal belongs to. In order to uniquely identify the terminal, both cell_id and terminal_id are required. More than one terminal per cell can have the same terminal name but they are distinguished by the conjunction of cell_id and terminal_id.

The attributes of the terminal table are set forth in Table 3.

TABLE 3

Key	Name	Type	Integrity	Description
1	cell_id	i4	>=0	cell identification number
2	terminal_id	i4	>=0	terminal identification number
	name	c20	alpha-numeric	terminal name
	type	c5	alpha-numeric	terminal type (vdd, gnd, signal)
	layer	c5	alpha-numeric	terminal layer (m1, poly, diff)
	direction	c1	l,r,t,b	terminal exit side
	x	i4		x coordinate of the terminal
	y	i4		y coordinate of the terminal
	width	i4		terminal width

The cellref table contains the hierarchical cell structure. Using this table, it is possible to learn if a cell has any parent, and therefore a data dependency. The attributes of the cellref table are set forth in Table 4:

TABLE 4

Key	Name	Type	Integrity	Description
1	parent_id	i4	>=0	cell id of the parent
2	child_id	i4	>=0	cell id of the child
3	instance_id	i4	>=0	instance id to distinguish cells
	x	i4		x coordinate of the instance
	y	i4		y coordinate of the instance
	MX	i2	Boolean	True if inst. is mirrored in X
	MY	i2	Boolean	True if inst. is mirrored in Y
	RO	i2	Boolean	Rotation variable in X
	R1	i2	Boolean	Rotation variable in Y

Technology refers to a particular process run of a IC foundry. For each entry in the technology table, there is a set of cells that uses the technology (called cell library). Each record in this table is a unique process run to not only within the foundry, but also within the corporation. The attributes of the technology table are set forth in Table 5 below:

TABLE 5

Key	Name	Type	Integrity	Description
1	technology_id	i4	>=0	id that identifies technology
	facility	c20	alpha-numeric	name and location of fab house
	device	c10	alpha-numeric	CMOS, bipolar, HCMOS, GaAs, etc.
	feature_size	i4	>0	channel width of a transistor
	no_metal	i2	>=0	number of metals used
	no_poly	i2	>=0	number of polysilicons used
	well_type	c10	alpha-numeric	N-Well, P-Well, etc.

Every standard frame used for a specific technology is described in the frame table. Frame table is related to the technology table by the key technology_id. Each frame must belong to one and only one technology, while each technology can have many frames. Each record in the frame table represents a frame. The attributes of the frame table are set forth in Table 6 below:

TABLE 6

Key	Name	Type	Integrity	Description
1	frame_id	i4	>=0	id that identifies each frame
2	technology_id	i4	>=0	id of corresponding technology
	no_pads	i2	>0	number of pads in the frame
	in_width	i4	>=0	width of usable space in frame
	in_height	i4	>=0	height of usable space in frame
	out_width	i4	>0	total frame width with pads
	out_height	i4	>0	total frame height with pads

The software subroutine library **62** includes for each microprocessor type, predefined software subroutines which correspond to the macro functions which can be implemented in software. Typical subroutines used in ISSC are standard I/O macros, data conversion macros, character conversion macros, and string manipulation macros.

Every macro function used in categorizing a cell in terms of its function is stored as a single record entry in a macro table contained within the macro function library **63**. Many cells within the cell library can be capable of performing the function of a macro. Each macro in the table is uniquely identified by the key macro_id. The attributes of the macro table are set forth in Table 7:

TABLE 7

Key	Name	Type	Integrity	Description
1	macro_id	i4	>=0	id that identifies each macro
	macro_name	c20	alpha-numeric	macro name describing function
	no_inputs	i2	>=0	number of inputs in function
	no_outputs	i2	>=0	number of outputs in function
	description	c80	sdl	description

CONGEN

CONGEN **27** used the STF file from BLATH **24** to design a PLA based system controller for the chip which generates control signals to enable the respective hardware cells or the microprocessor. The STF file is translated to Boolean equations for PLA logic that implements next state codes and output functions. A unique code is assigned for each state by applying heuristic rules for state assignment. The output of CONGEN is an input parameter file for automatic PLA layout generation.

SOFTCOM

SOFTCOM **70** takes as input the STF file **26** created by BLATH **24** and generates an assembly program which is the monitor program for the microprocessor. Some of the variables of this program are:

NUM_SHARED is the number of shared variables between software and hardware

REG_ARRAY[0.. NUM_SHARED] is a external register array.

NUM_SUB is the number of software routines in the system.

SHARED_VARS is an array of memory locations at which the shared variables are stored.

change_flag: associated with each shared variable is a flag which is set if that variable's value is changed.

The program format is as follows:

```
1)  set change_flag for shared variables to false.
2)  get subroutine code from REG_ARRAY[0]
3)  if code = = 1 then call subroutine 1
    if code = = 2 then call subroutine 2
    . . .
    . . .
    if code = = NUM_SUB then call subroutine_num_sub
4)  for i = 1 to NUM_SHARED do
    {
        if change_flag set then
        move SHARED_VARS[i] TO REG_ARRAY[i]
    }
5)  transfer control to the system controller
```

Design Example

This example illustrates the design approach employed in designing an application specific integrated circuit which does simple computer graphics operations such as translation, mirror image of X axis or Y axis, and rotation. All of those operations are involved in matrix multiplication with the transformation axis. The matrix multiplication will be implemented in hardware to speed up the process. FIG. **5** is a flowchart illustrating a first software portion (software portion #1) of the foregoing program which is to be processed by the microprocessor subsystem within the target ASIC. FIG. **6** is a continuation of the flowchart of FIG. **5** and illustrates the remaining portion of software portion #1 as well as software portion #2 to be processed by the microprocessor subsystem within the target ASIC, and additionally shows the non-microprocessor hardware subsystem within the target ASIC.

FIG. **7** is a block schematic diagram similar to the general diagram of FIG. **3** but showing the components which would be utilized in the above-noted specific example. To avoid repetitive description, elements in FIG. **7** which correspond to elements previously described in connection with FIG. **3** will be identified by the same reference characters with prime notation (') added. The microprocessor subsystem **80'** contains CPU **81'** and RAM **82'** (main memory). The execution code of software portions #1 and #2 (**82***a*, **82***b*) are stored in the main memory of microprocessor subsystem. The data segment **82***c* contains the data to be processed by both CPU and non-microprocessor hardware subsystem. The data in data segment **82***c* are sent to data registers via the system data bus **83'**. The communication between microprocessor subsystem **80'** and hardware subsystem **90'** is coordinated by using interrupts. After software portion #1 (**82***a*) is executed, microprocessor subsystem **80'** reads the status registers to see if hardware subsystem **90'** is busy. If hardware subsystem **90'** is idle, its status registers are set, and the hardware subsystem **90'** is activated.

When hardware subsystem **90'** finishes the task, it generates an interrupt signal to microprocessor subsystem **80'**. The interrupt is serviced by interrupt handler **82***d*. The main tasks of the interrupt handler include:

push contents of microprocessor subsystem **80'** registers into stack.

transfer data from hardware subsystem **90'** to microprocessor subsystem **80'**.

reset the status register of hardware subsystem **90'**.

restore the contents of microprocessor subsystem **80'** registers from stack.

continue the execution of software portion #2 (**82***b*).

The execution flow of the entire flowchart (including software and hardware portions) is as follows:

1. Microprocessor subsystem **80'** executes the codes in software portion #(**82***a*).
2. Microprocessor subsystem **80'** activates hardware subsystem **80'** by overwriting the status registers, then sends the data to data registers.
3. Hardware subsystem **90'** processes the data in data registers
4. After hardware subsystem **90'** processed the data, it sends interrupt signal to microprocessor subsystem **80'**.
5. The interrupt handler **82***d* of microprocessor subsystem **80'** handles the interrupt, transfers the data from data registers back to microprocessor subsystem, and resets the status registers. After the interrupt is processed, the execution flow is returned to software portion #2.
6. Microprocessor subsystem **80'** executes the codes in software portion #2.
7. Done.

As shown in FIG. **6**, states **24–35** and conditions **9–11** are implemented in hardware subsystem to perform multiply/accumulate operation. A multiplier and an adder are used in state **28** to implement the multiply/accumulate operations. The blocks indicated as state **24**, state **25**, state **27**, state **29**, state **32**, and state **34** use counters i, j, k, which have increment and clear capabilities. In state **26** and state **31**, temp is used as a temporary register to store the intermediate result. State **30**, state **33**, and state **35** use a comparator. The result of the comparison will affect the control flow which specified in cond**9**, cond**10**, and cond**11**.

The execution flow of hardware subsystem is as follows:

1. Microprocessor subsystem sends a start signal to hardware subsystem via system control bus.
2. All the data are loaded to data registers via system data bus.
3. Loop counters i, j, and k are reset to zero initially. Register temp is also reset to zero.
4. Multiply/Accumulate operation.
5. Increment the loop counters and check if the end of the loops. If it is not end of the loops, go to step 3; else exits the loops.
6. The result of multiply/accumulate operation is stored in the output register that is connected to system data bus.
7. An interrupt signal is sent to the microprocessor subsystem via system control bus.

While the invention has been described herein with reference to a specific embodiment, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications and applications may occur to those skilled in the art without departing from the true spirt and scope of the invention as defined by the appended claims.

That which is claimed is:

1. A computer-aided design system for designing an application specific integrated circuit from architecture independent functional specifications for the integrated circuit, whose intended function is implemented by both a hardware subsystem including hardware elments on the integrated circuit and by a software subsystem including software instructions executed by a microprocessor on the integrated circuit, comprising

input means operable by a user for defining architecture independent functional specifications for the application specific integrated circuit wherein at least one of said architecture independent functional specifications is free of indication that an intended function of said integrated circuit is implanted on said hardware subsystem or said software subsystem, and

computer operated means for translating the architecture independent functional specifications into an architecture specific structural level definition of the integrated circuit, said computer operated translating means comprising

means defining software instructions of the software subsystem from the architecture independent functional specifications, at least one of which is free of indication that an intended function of said inte-

grated circuit is implemented on said hardware subsystem or said software subsystem,

means defining a microprocessor for executing the software instructions of the software subsystem from the architecture independent functional specifications, at least one of which is free of indication that an intended function of said integrated circuit is implemented on said hardware subsystem or said software subsystem,

means defining hardware elements for executing the hardware functions of the hardware subsystem from the architecture independent functional specifications, at least one of which is free of indication that an intended function of said integrated circuit is implemented on said hardware subsystem or said software subsystem, and

means defining interconnections between the microprocessor for executing the software instructions of the software subsystem and the hardware elements of the hardware subsystem.

2. The system as defined in claim 1 wherein said architecture independent functional specifications are comprised of a series of operations, and wherein said computer operated means comprises

a cell library defining a set of available integrated circuit hardware cells for performing said operations;

a software subroutine library defining a set of available software subroutines for performing said operations; and

selection means for selecting from said cell library or from said software subroutine library for each operation specified by said specification input means, appropriate hardware cells or software subroutines for performing the operations specified by the functional specifications.

3. The system as defined in claim 2 wherein said selection means comprises an expert system including a knowledge base containing rules for selecting hardware cells from said cell library and software subroutines from said software subroutine library, and inference engine means for selecting appropriate hardware cells or software subroutines from said libraries in accordance with the rules of said knowledge base.

4. A computer-aided design system for designing an application specific integrated circuit from architecture independent functional specifications for the integrated circuit whose intended function is implemented by both a hardware subsystem including hardware elements on the integrated circuit and by a software subsystem including software instructions executed by a microprocessor on the integrated circuit, comprising

a macro library defining a set of architecture independent operations comprised of actions and conditions wherein at least one of said architecture independent operations is free of indication that the operation is implemented on said hardware subsystem or said software subsystem;

specification input means operable by a user for defining architecture independent functional specifications for the integrated circuit wherein at least one of said architecture independent functional specifications is free of indication that an intended function of said integrated circuit is implemented on said hardware subsystem or said software subsystem, said architecture independent functional specifications being comprised of a series of architecture independent operations which implement the intended function of the application specific integrated circuit wherein at least one of said operations is free of indication that an intended function of the integrated circuit is implemented or said hardware subsystem or said software subsystem, said specification input means including means to permit the user to specify for each operation in the defined series of operations a macro selected from said macro library;

a hardware cell library defining a set of available integrated circuit hardware cells for performing operations defined in said macro library wherein at least one of said operations is free of indication that an intended function of the integrated circuit is implemented on said hardware subsystem or said software subsystem;

a software subroutine library defining a set of available software subroutines for performing operations defined in said macro library wherein at least one of said operations is free of indication that an intended function of the integrated circuit is implemented on said hardware subsystem or said software subsystem;

selection means for selecting from said hardware cell library or from said software subroutine library for each macro specified by said specification input means, appropriate hardware cells or software subroutines for performing the operation defined by the specified macro wherein at least one of said operations is free of indication that an intended function of the integrated circuit is implemented on said hardware subsystem or said software subsystem, said selection means comprising an expert system including a knowledge base containing rules for selecting hardware cells from said hardware cell library and software subroutines from said software subroutine library, and inference engine means for selecting appropriate hardware cells or software subroutines from said libraries in accordance with the rules of said knowledge base.

5. The system as defined in claim 4 wherein said specification input means includes means to enable the user, when specifying a macro selected from said macro library, to define whether the macro function is to be implemented in hardware or in software.

6. The system as defined in claim 4 additionally including design constraint input means operable by a user for defining design constraints for the application specific integrated circuit.

7. The system as defined in claim 6 wherein said design constraint input means includes means to enable the user to define one or more design constraints for the application specific integrated circuit selected from the group consisting of annual volume, speed, size, pin count, packaging type, power consumption, and thermal stability.

8. The system as defined in claim 7 wherein said knowledge base additionally contains design style rules for selecting, based upon the design constraints, an optimum design style for implementation of the hardware subsystem.

9. The system as defined in claim 8 wherein said design style is selected from the group consisting of programmable logic device (PLD), gate array, standard cell, and macro cell.

10. The system as defined in claim 6 wherein said design constraint input means includes means to enable the user to select, from a set of predefined available

standard microprocessors, the particular microprocessor type to be utilized by the software subsystem.

11. The system as defined in claim **10** wherein said software subroutine library includes sets of software subroutines corresponding to each available standard microprocessor type.

12. The system as defined in claim **6** wherein said knowledge base additionally contains design constraint rules for selecting appropriate hardware cells or software subroutines based upon the design constraints defined by said design constraint input means.

13. The system as defined in claim **4** wherein said specification input means comprises flowchart editor means for creating a flowchart having elements representing said series of actions and conditions.

14. The system as defined in claim **4** wherein said specification input means comprises means for receiving user input of a list defining the series of actions and conditions which implement the function of the application specific integrated circuit.

15. The system as defined in claim **4** wherein said selection means comprises means for generating a netlist defining the microprocessor to be utilized by the software subsystem and the hardware cells to be utilized by the hardware subsystem, and interconnection information for interconnecting the microprocessor and the hardware cells to perform the intended function of the application specific integrated circuit.

16. The system as defined in claim **15** wherein said selection means additionally includes control generator means for generating a controller and control paths for controlling operation of the microprocessor and the hardware cells selected by said cell selection means.

17. The system as defined in claim **15** additionally including mask data generator means for generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

18. The system as defined in claim **4** additionally including software compiler means for generating software operable by the microprocessor for performing functions of the software subsystem.

19. A computer-aided design system for designing an application specific integrated circuit from a flowchart defining architecture independent functional requirements of the integrated circuit, whose intended function is implemented by both a hardware subsystem including hardware elements on the integrated circuit and by a software subsystem including software instructions executed by a microprocessor on the integrated circuit, comprising

a computer system;

a macro library, stored in said computer system, defining a set of possible architecture independent actions and conditions wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

flowchart editor means, executing on said computer system, operable by a user for creating a flowchart having elements representing said architecture independent actions and conditions wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem, said flowchart editor means, executing on said computer system, including macro specification means, executing on said computer system, for permitting the user to specify for each architecture independent action or condition represented in the flowchart a macro selected from said macro library;

a hardware cell library, stored in said computer system, defining a set of available integrated circuit hardware cells for performing the architecture independent actions and conditions defined in the macro library wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

a software subroutine library, stored in said computer system, defining a set of available software subroutines for performing the architecture independent actions and conditions defined in the macro library wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem; and

selection means, executing on said computer system, for selecting from said hardware cell library or from said software subroutine library for each macro specified by said flowchart editor means, executing on said computer system, appropriate hardware cells or software subroutines for performing the architecture independent action or condition defined by the specified macro, and for generating a netlist defining the microprocessor to be utilized by the software subsystem and the hardware cells to be utilized by the hardware subsystem, and interconnection information for interconnecting the microprocessor for executing the software subroutines and the hardware cells to perform the intended function of the application specific integrated circuit.

20. The system as defined in claim **19** wherein said selection means comprises a knowledge base containing rules for selecting hardware cells from said cell library and software subroutines from said software subroutine library, and inference engine means for selecting appropriate hardware cells or software subroutines from said libraries in accordance with the rules of said knowledge base.

21. A computer-aided design process for designing, in a computer system, an application specific integrated circuit from architecture independent functional specifications for the integrated circuit, whose intended function is implemented by both a hardware subsystem including hardware elements on the integrated circuit and by a software subsystem including software instructions executed by a microprocessor on the integrated circuit, comprising the following steps performed by said computer system:

storing in a macro library in said computer system, a set of definitions of possible architecture independent actions and conditions wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

storing in a hardware cell library in said computer system, data describing a set of available integrated circuit hardware cells for performing the architecture independent actions and conditions wherein at

least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

storing in a software subroutine library in said computer system, data describing a set of software subroutines for performing the architecture independent actions and conditions wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

storing in an expert system knowledge base in said computer system, a set of rules for selecting hardware cells or software subroutines to perform the architecture independent actions and conditions wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions which will implement the intended function of the application specific integrated circuit wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

specifying for each described architecture independent action and condition of the series one of said stored definitions from the macro library stored in said computer system which corresponds to the desired architecture independent action or condition to be performed; and

selecting from said hardware cell library stored in said computer system or from said software subroutine library stored in said computer system for each of the specified definitions, appropriate integrated circuit hardware cells of software subroutines for performing the desired function of the application specific integrated circuit.

22. A process as defined in claim 21 wherein said step of selecting a hardware cell or a software subroutine comprises applying to the specified definition of the action or condition to be performed, a set of cell selection rules and software subroutine selection rules stored in the knowledge base in said computer system.

23. A process as defined in claim 21 including the further step performed by said computer system of generating a netlist defining the microprocessor and the hardware cells which are needed to perform the desired function of the integrated circuit and the connections therebetween.

24. A process as defined in claim 23 including the further step of generating from said netlist the mask data required to produce an integrated circuit having the specified functional requirements.

25. A computer-aided knowledge based design process for designing in a computer system an application specific integrated circuit from architecture independent functional specifications for the integrated circuit, whose intended function is implemented by both a hardware subsystem including hardware elements on the integrated circuit and by a software subsystem including software instructions executed by a microprocessor on the integrated circuit, comprising the following steps performed by said computer system:

storing in a macro library in said computer system, a set of macros defining possible architecture independent actions and conditions wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

storing in a hardware cell library in said computer system, data describing a set of available integrated circuit hardware cells for performing the architecture independent actions and conditions wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

storing in a software subroutine library in said computer system, data describing a set of available software subroutines for performing the architecture independent actions and conditions wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

storing in a knowledge base in said computer system, a set of rules for selecting hardware cells from said hardware cell library and software subroutines from said software subroutine library to perform the architecture independent actions and conditions defined by the stored macros wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

describing for a proposed application specific integrated circuit a series of architecture independent actions and conditions which carry out the function to be performed by the hardware cells and software subroutines of the integrated circuit wherein at least one of said architecture independent actions and conditions is free of indication that the action or condition is implemented on said hardware subsystem or said software subsystem;

specifying for each described action and condition of said series a macro selected from the macro library stored in said computer system which corresponds to the action or condition; and

applying rules of said knowledge base stored in said computer system to the specified macros to select from said hardware cell library stored in said computer system and from said software subroutine library stored in said computer system, the hardware cells and software subroutines required for performing the desired function of the application specific integrated circuit.

26. A process as defined in claim 25 wherein said step of describing a series of actions and conditions comprises creating a flowchart comprised of elements representing actions and conditions.

27. A process as defined in claim 25 wherein said step of specifying a macro selected from the macro library further includes defining whether the macro function is to be implemented in hardware or in software.

28. A process as defined in claim 27 including the step of generating software code for use by the microprocessor in performing the macro functions which are specified to be implemented in software.

29. A process as defined in claim 25 including defining one or more design constraints for the application

specific integrated circuit selected from the group consisting of annual volume, speed, size, pin count, packaging type, power consumption, and thermal stability.

30. A process as defined in claim **29** wherein said knowledge base additionally contains design style rules for selecting, based upon the design constraints, an optimum design style for implementation of the hardware subsystem, and wherein said step of applying the rules of the knowledge base includes selecting, based upon the design constraints, an optimum design style for implementation of the hardware subsystem.

31. A process as defined in claim **30** wherein said design style is selected from the group consisting of programmable logic device (PLD), gate array, standard cell, and macro cell.

32. A process as defined in claim **25** wherein said step of specifying a macro selected from the macro library further includes indicating, for at least certain ones of the specified macros, that the selection of whether the macro function is to be implemented in hardware or in software is to be made by the computer system, and

wherein said knowledge base additionally contains implementation rules for determining the optimum implementation of macro functions either in the hardware subsystem or in the software subsystem, and

wherein said step of applying the rules of the knowledge base includes determining, based upon the implementation rules in said knowledge base, the optimum implementation of the macro function either in the hardware subsystem or the software subsystem.

* * * * *

6

United States Patent [19]

Dunn

[11] **Patent Number: 4,656,603**

[45] **Date of Patent: Apr. 7, 1987**

[54] **SCHEMATIC DIAGRAM GENERATING SYSTEM USING LIBRARY OF GENERAL PURPOSE INTERACTIVELY SELECTABLE GRAPHIC PRIMITIVES TO CREATE SPECIAL APPLICATIONS ICONS**

[75] Inventor: **Robert M. Dunn**, Woodbridge, Conn.

[73] Assignee: **The Cadware Group, Ltd.**, New Haven, Conn.

[21] Appl. No.: **585,535**

[22] Filed: **Mar. 1, 1984**

[51] **Int. Cl.**[4] **G06F 3/153**; G06F 15/40

[52] **U.S. Cl.** **364/900**; 364/488; 364/521; 340/721; 340/747

[58] **Field of Search** ... 364/200 MS File, 900 MS File, 364/300, 513–515, 521; 340/721, 747, 750, 799

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,315,315	2/1982	Kossiakoff	364/300
4,455,619	6/1984	Masul et al.	364/900
4,546,435	10/1985	Herbert et al.	364/300
4,555,772	11/1985	Stephens	364/900

FOREIGN PATENT DOCUMENTS

0100798	2/1984	European Pat. Off.	364/521
2941824	4/1980	Fed. Rep. of Germany	364/300

OTHER PUBLICATIONS

D. Laidlaw, *Graphic Entity Transformation & Manipulation*, IBM Technical Disclosure Bull., (vol. 21, No. 3, Aug. 1978), pp. 1234–1243.

P. Groner, *Computer Aided Design of VLSI Saves Man–Hours, Reduces Errors*, Control Engineering (Apr. 1981), pp. 55–57.

R. A. Armstrong, *Applying CAD to Gate Arrays Speeds 32-Bit Mini Computer Design*, Electronics (Jan. 13, 1981), pp. 167–173.

D. R. Fullenwider et al, *Computer Graphics & the Practice of Architecture*, IEEE Computer Graphics (Oct. 1981) pp. 19–26.

Primary Examiner—Archie E. Williams, Jr.
Attorney, Agent, or Firm—Barry R. Lipsitz

[57] **ABSTRACT**

An interactive rule based system enables problem solutions to be generated in schematic diagram form. A methodology designer selects and arranges graphic primitives using a graphics terminal to create a library of icons. Under control of a computer processor, the methodology designer is prompted to identify, by way example, parameters for using each icon. The system generates and stores a specific set of rules pertaining to the use of each icon on the basis of the parameters identified. The stored rules are cross-referenced to the icon to which they pertain, so that whenever the icon is selected by a problem solving user for use in building a problem solution, the rules pertaining thereto will be accessed and applied. A methodology designer can also select and concatenate functions to each other and to icons to create more complex functions for use in building problem solutions. New functions can also be created in the form of truth tables which establish a transfer function across an icon. By accessing and selecting icons and functions created by a methodology designer, a problem solving user can build a solution to a problem by graphically coupling the icons and functions together on a chart.

19 Claims, 16 Drawing Figures

Microfiche Appendix Included
(3 Microfiche, 200 Pages)

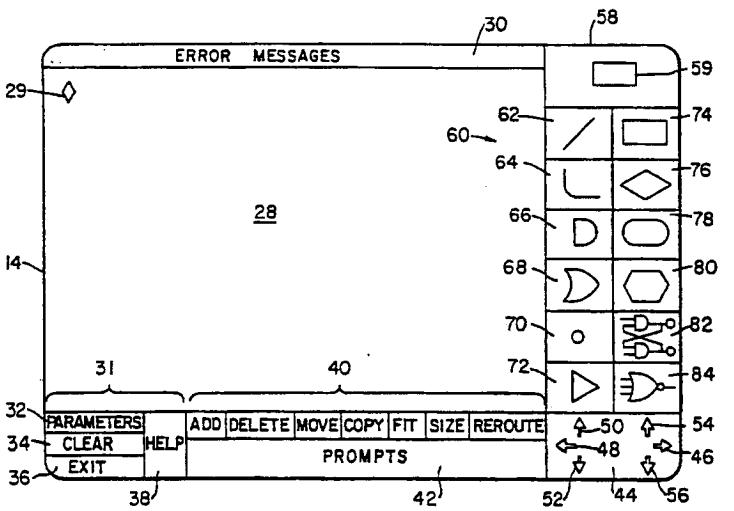


FIG. 1

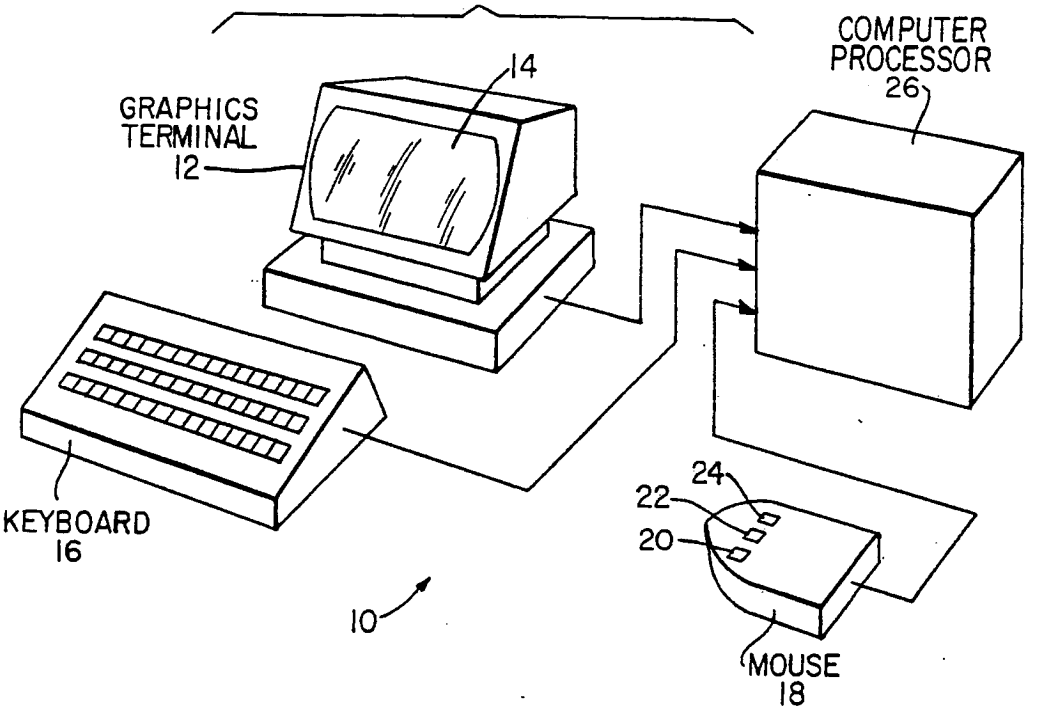
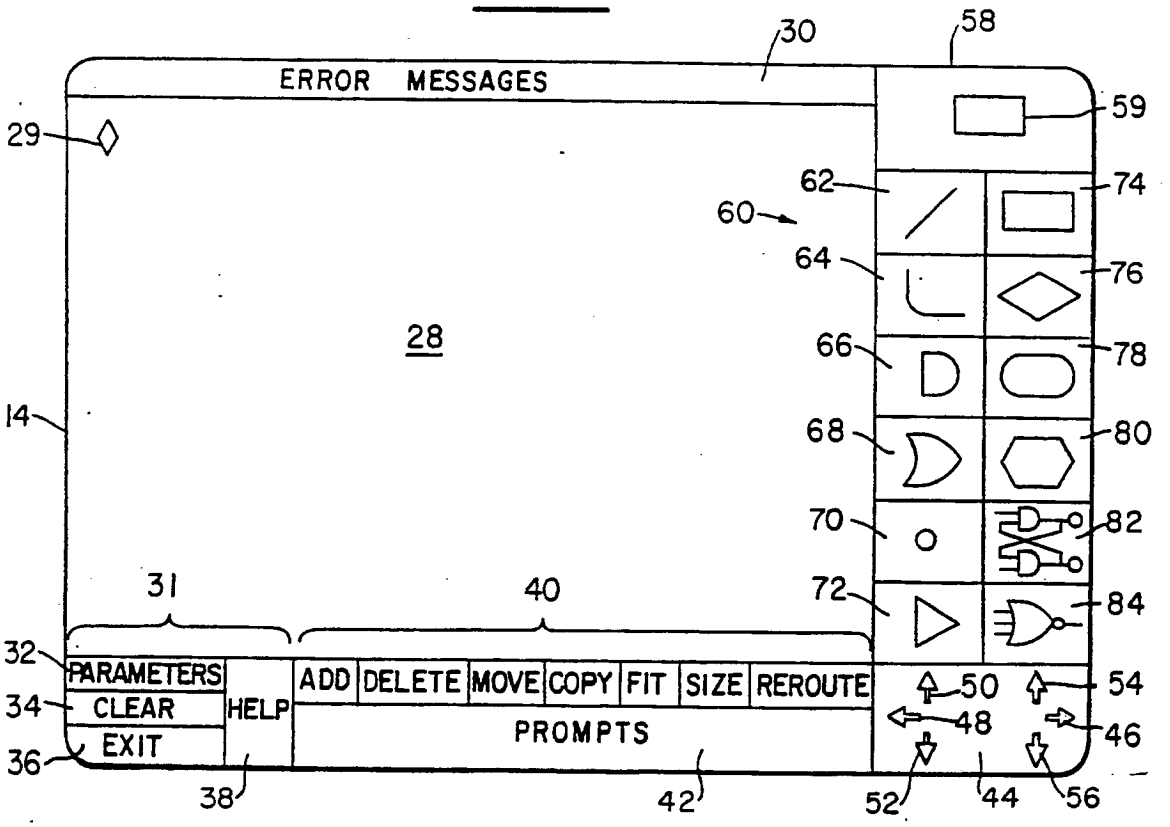


FIG. 2



DEF017457

FIG. 3

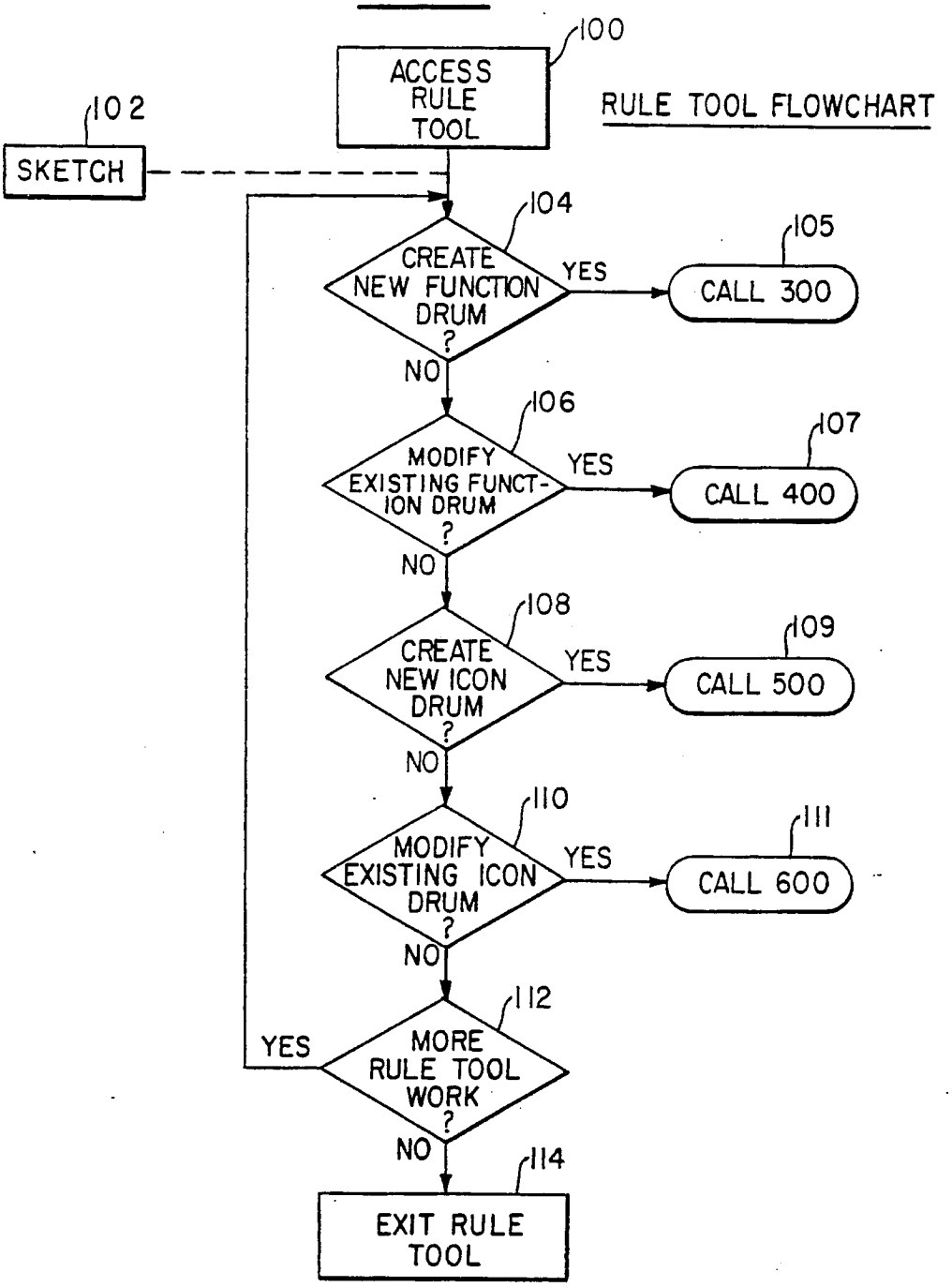
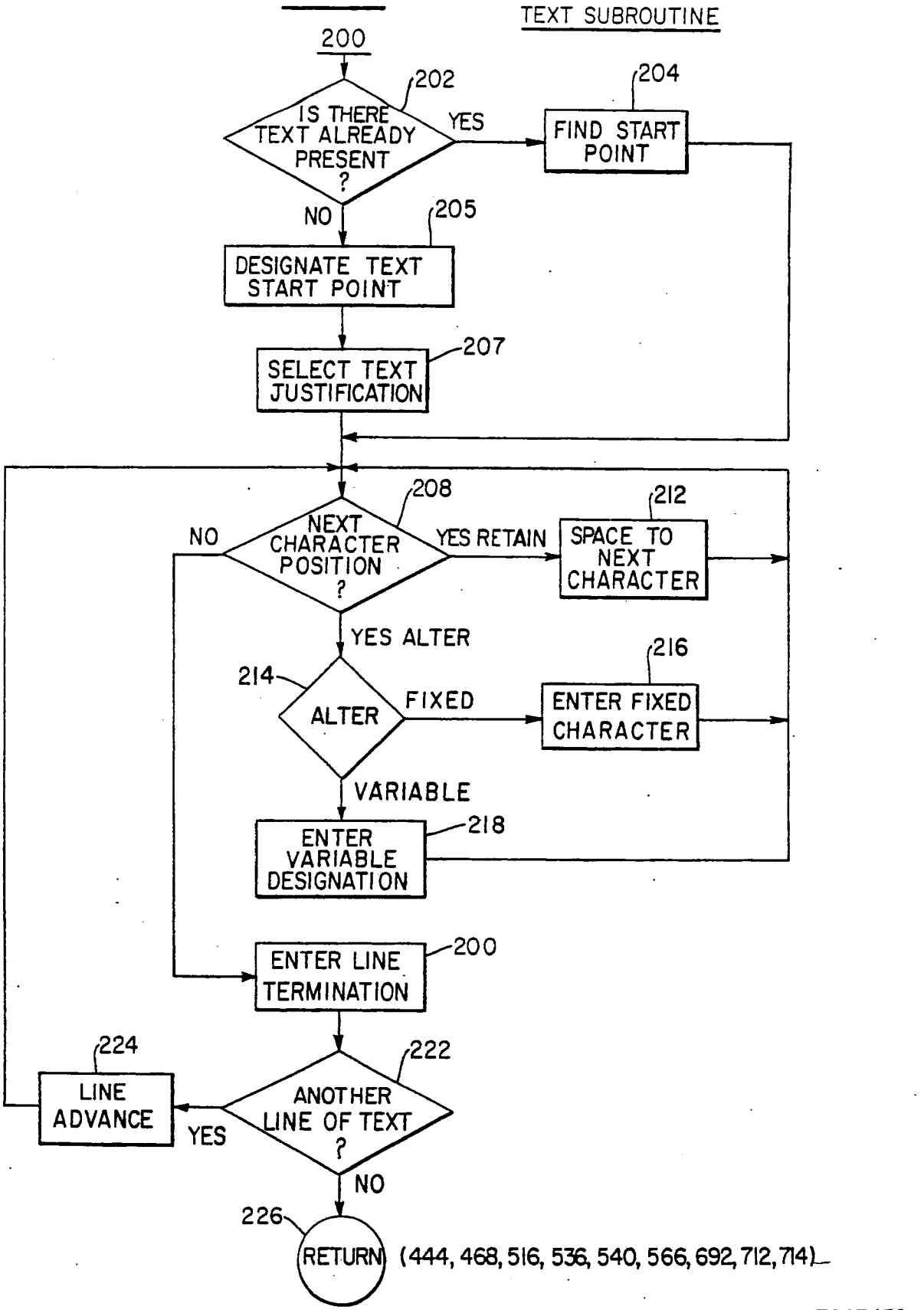

100
ACCESS RULE TOOL
RULE TOOL FLOWCHART
102
SKETCH
104
CREATE NEW FUNCTION DRUM ?
YES
105
CALL 300
NO
106
MODIFY EXISTING FUNCT-ION DRUM ?
YES
107
CALL 400
NO
108
CREATE NEW ICON DRUM ?
YES
109
CALL 500
NO
110
MODIFY EXISTING ICON DRUM ?
YES
111
CALL 600
NO
112
MORE RULE TOOL WORK ?
YES
NO
114
EXIT RULE TOOL

FIG. 4



DEF017459

FIG. 5

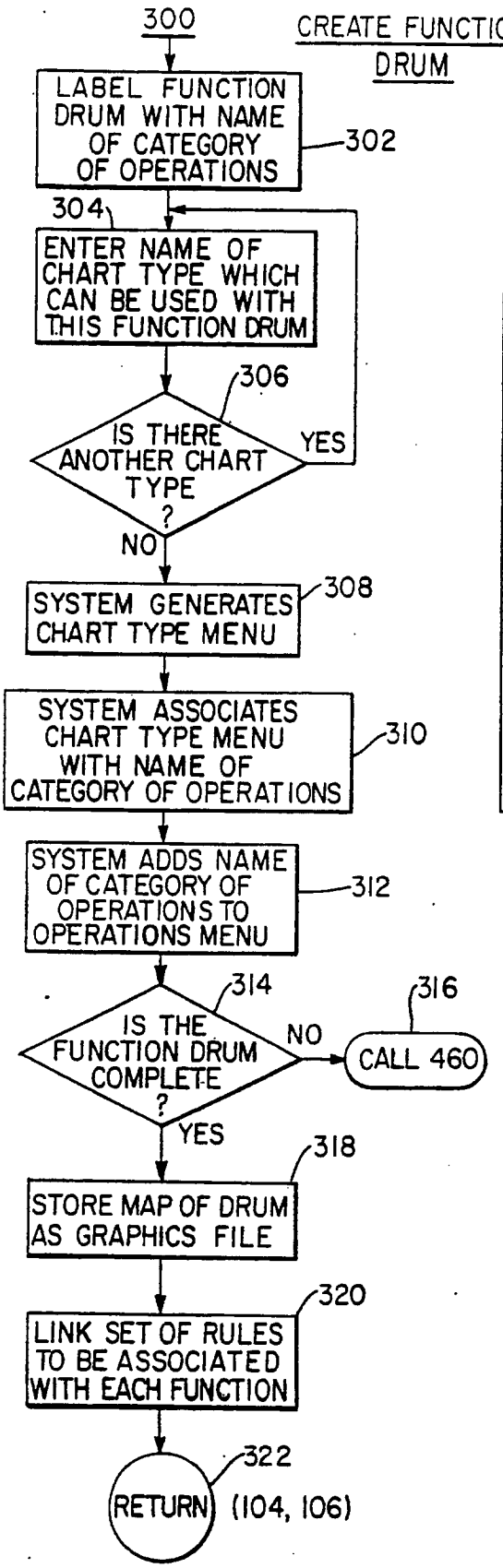
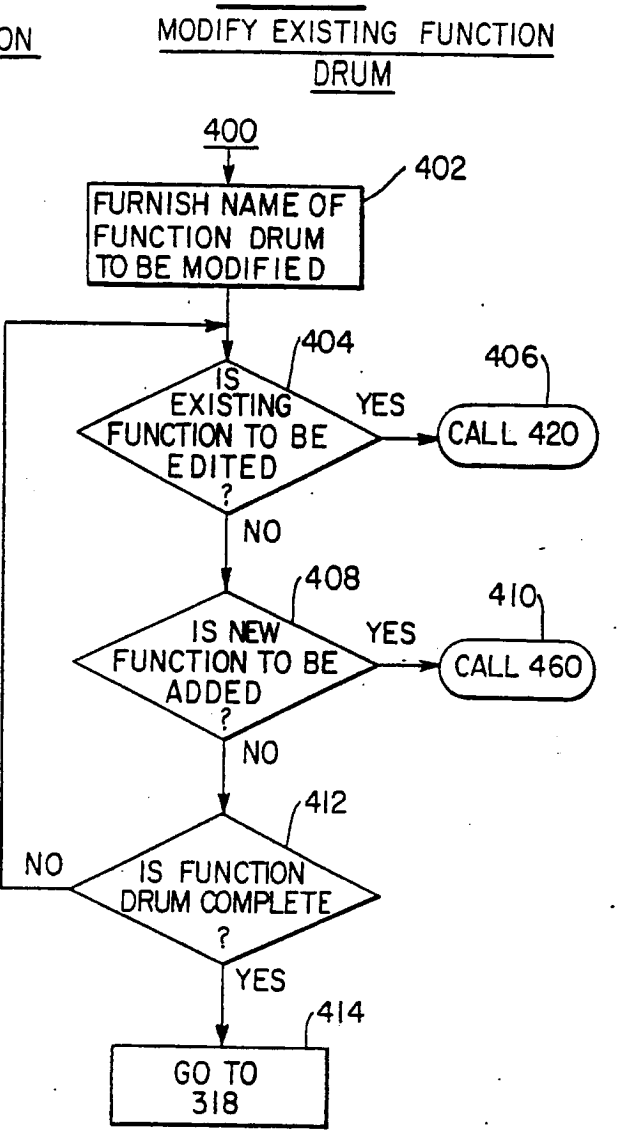


FIG. 6



DEF017460

FIG. 7

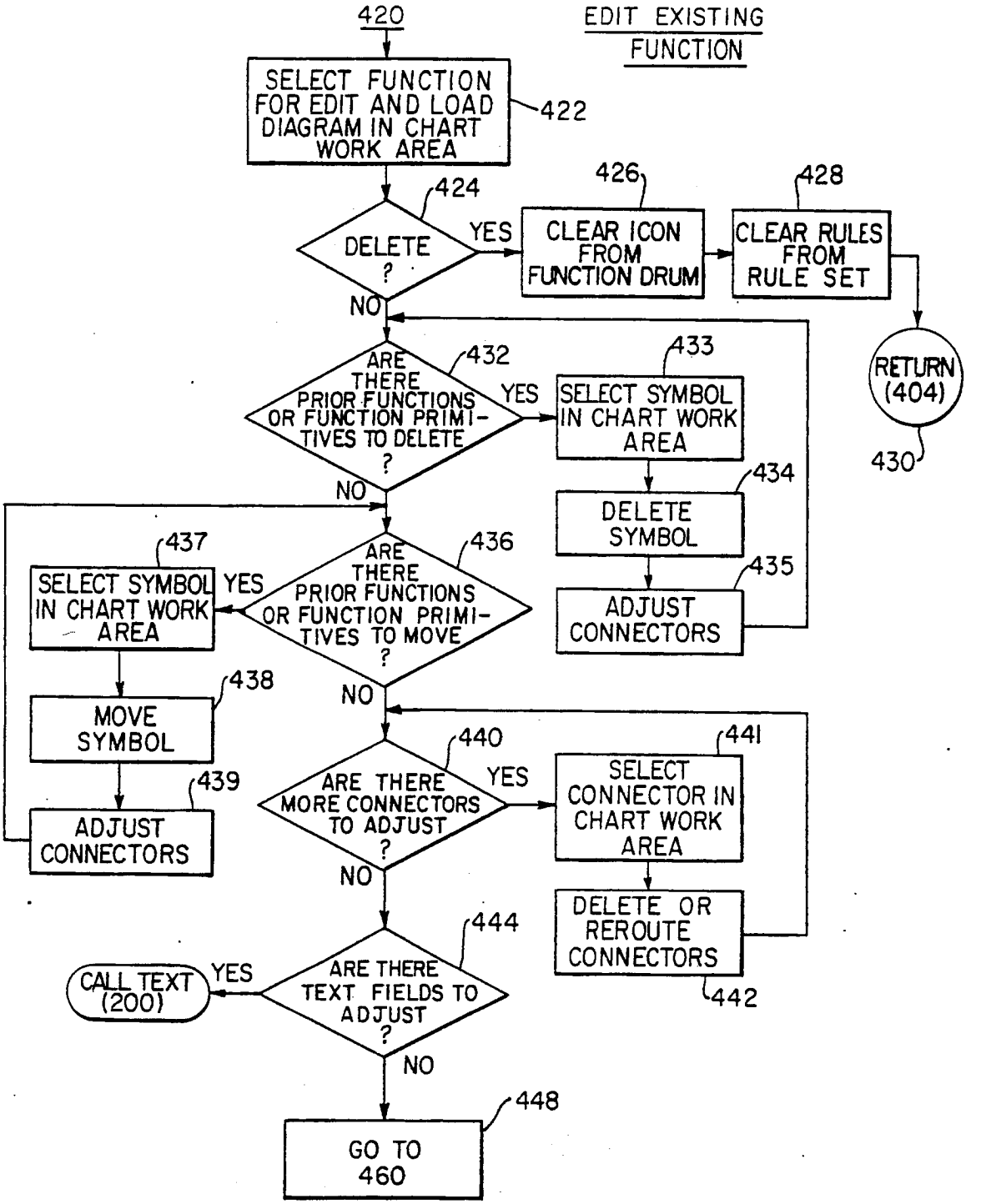

420
EDIT EXISTING FUNCTION
SELECT FUNCTION FOR EDIT AND LOAD DIAGRAM IN CHART WORK AREA
422
424
DELETE ?
YES
426
CLEAR ICON FROM FUNCTION DRUM
428
CLEAR RULES FROM RULE SET
NO
RETURN (404)
430
ARE THERE PRIOR FUNCTIONS OR FUNCTION PRIMITIVES TO DELETE ?
432
YES
433
SELECT SYMBOL IN CHART WORK AREA
434
DELETE SYMBOL
435
ADJUST CONNECTORS
NO
ARE THERE PRIOR FUNCTIONS OR FUNCTION PRIMITIVES TO MOVE ?
436
YES
437
SELECT SYMBOL IN CHART WORK AREA
438
MOVE SYMBOL
439
ADJUST CONNECTORS
NO
ARE THERE MORE CONNECTORS TO ADJUST ?
440
YES
441
SELECT CONNECTOR IN CHART WORK AREA
DELETE OR REROUTE CONNECTORS
442
NO
ARE THERE TEXT FIELDS TO ADJUST ?
444
YES
CALL TEXT (200)
NO
448
GO TO 460

FIG. 8

ADD NEW FUNCTION

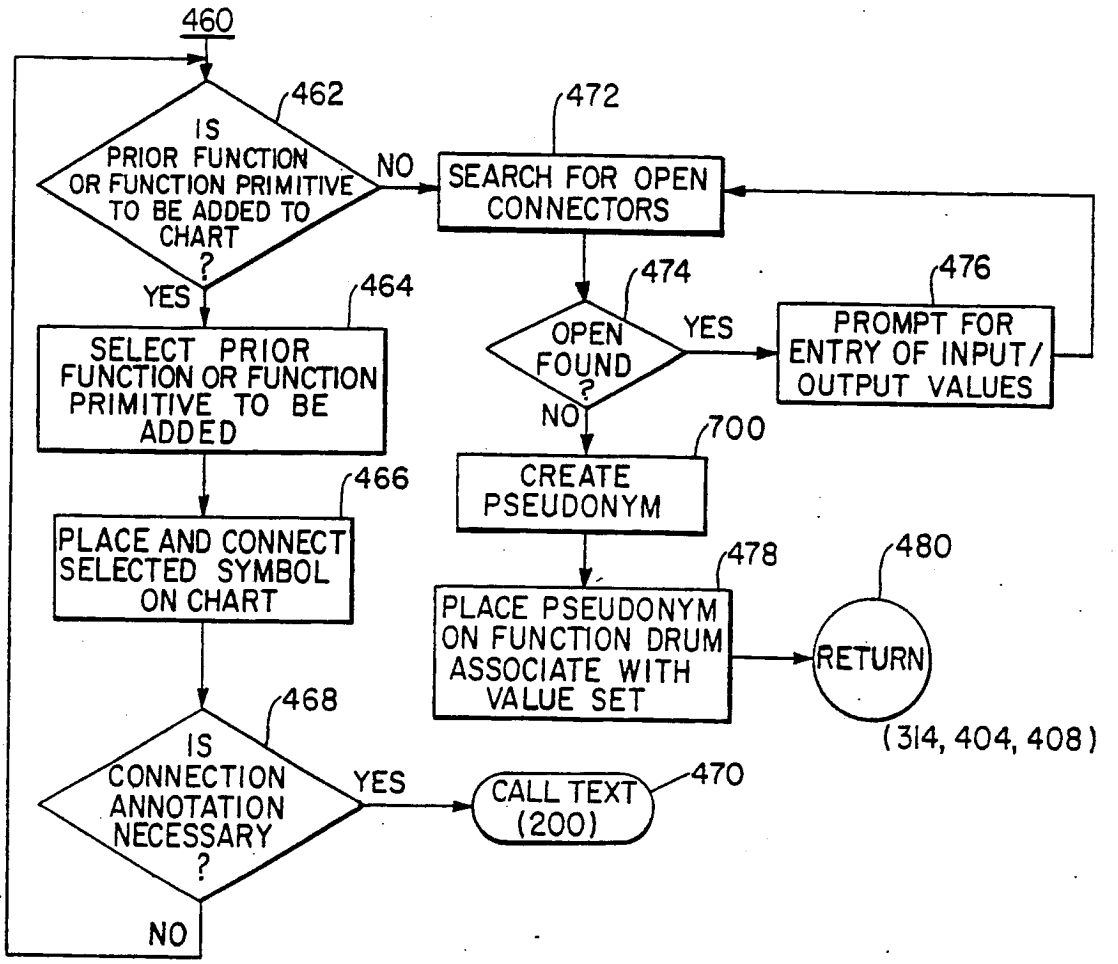
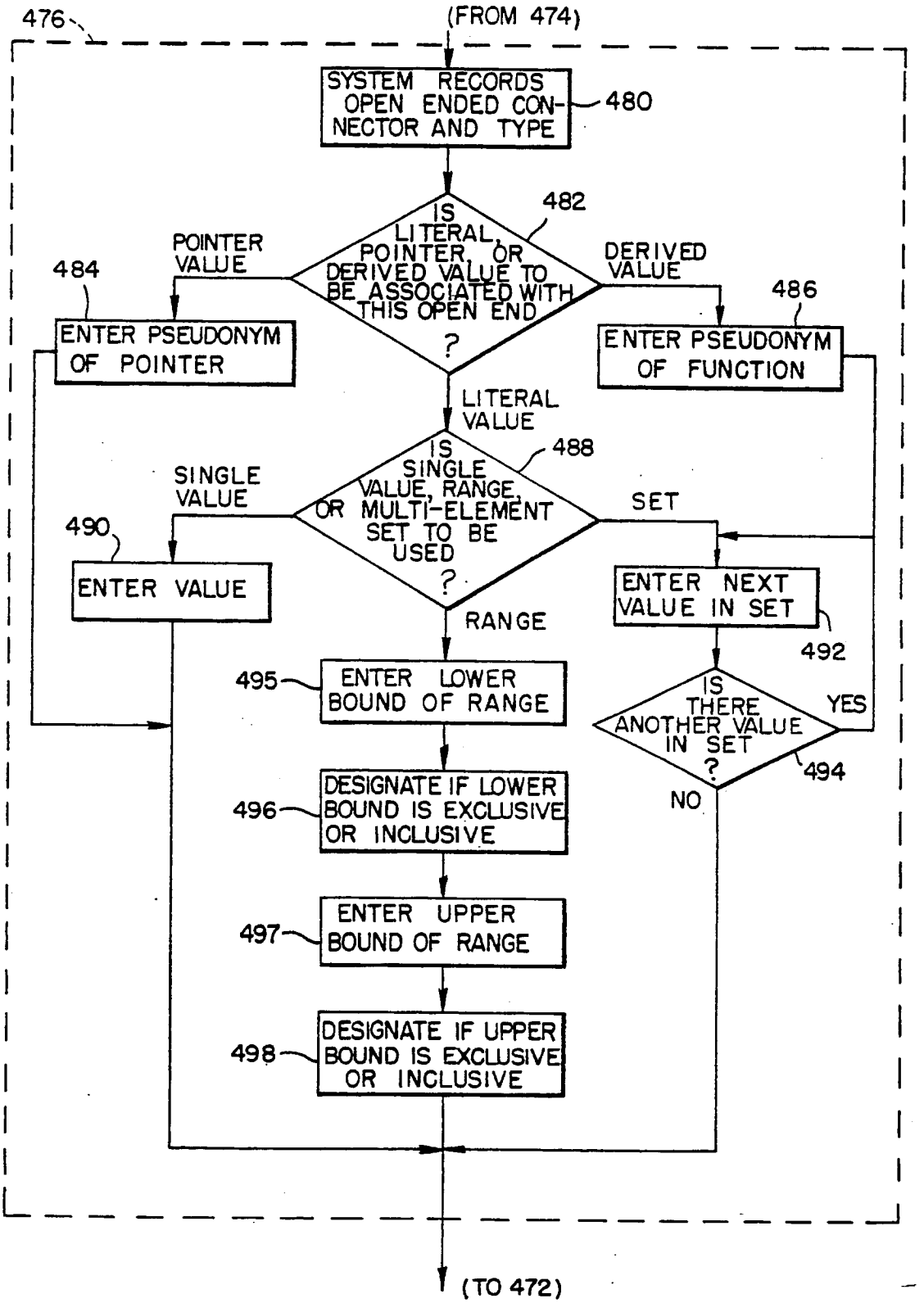


FIG. 9.

PROMPTING OF OPEN ENDED CONNECTORS



DEF017463

FIG. 10.

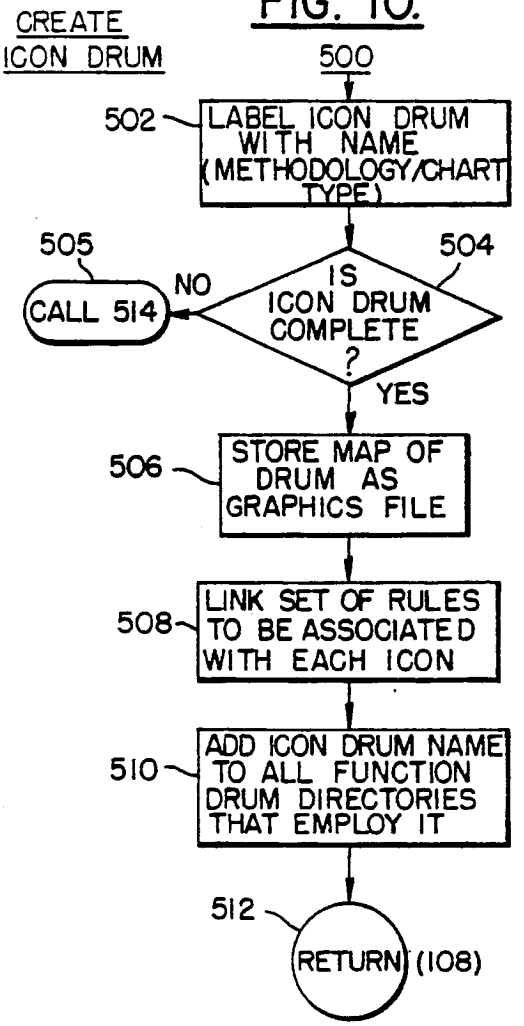

CREATE ICON DRUM
500
502
LABEL ICON DRUM WITH NAME (METHODOLOGY/CHART TYPE)
504
IS ICON DRUM COMPLETE ?
NO
505
CALL 514
YES
506
STORE MAP OF DRUM AS GRAPHICS FILE
508
LINK SET OF RULES TO BE ASSOCIATED WITH EACH ICON
510
ADD ICON DRUM NAME TO ALL FUNCTION DRUM DIRECTORIES THAT EMPLOY IT
512
RETURN (108)

FIG. 12.

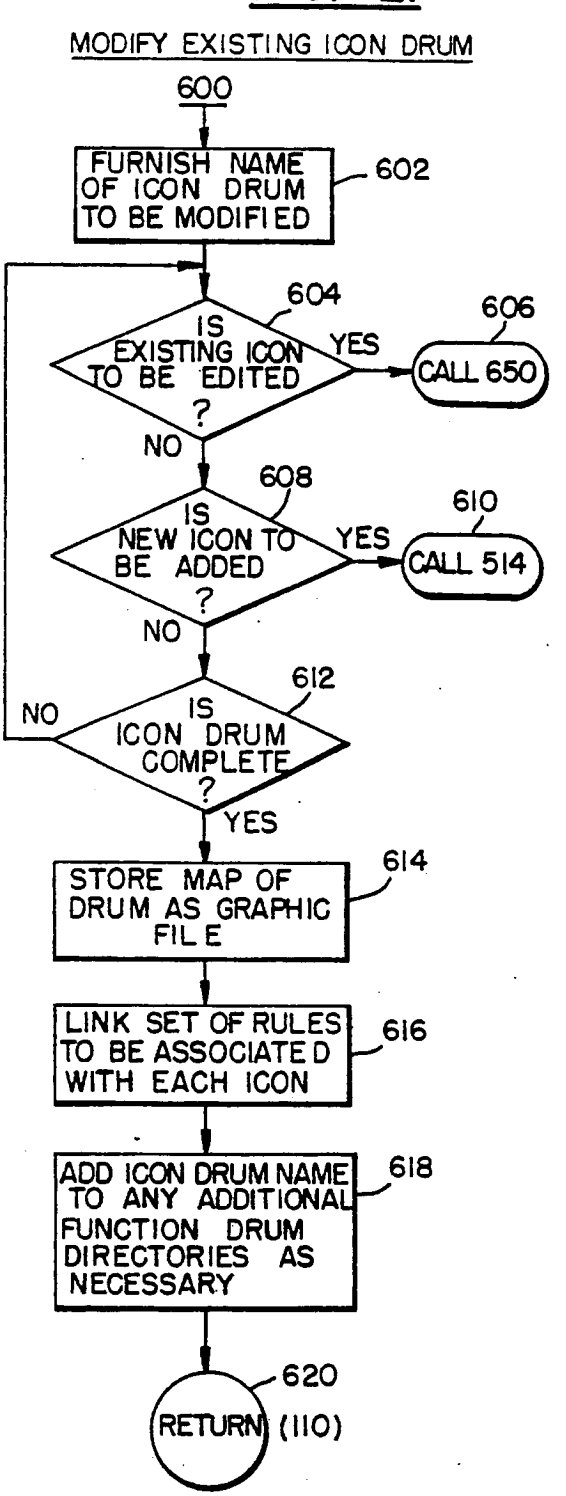
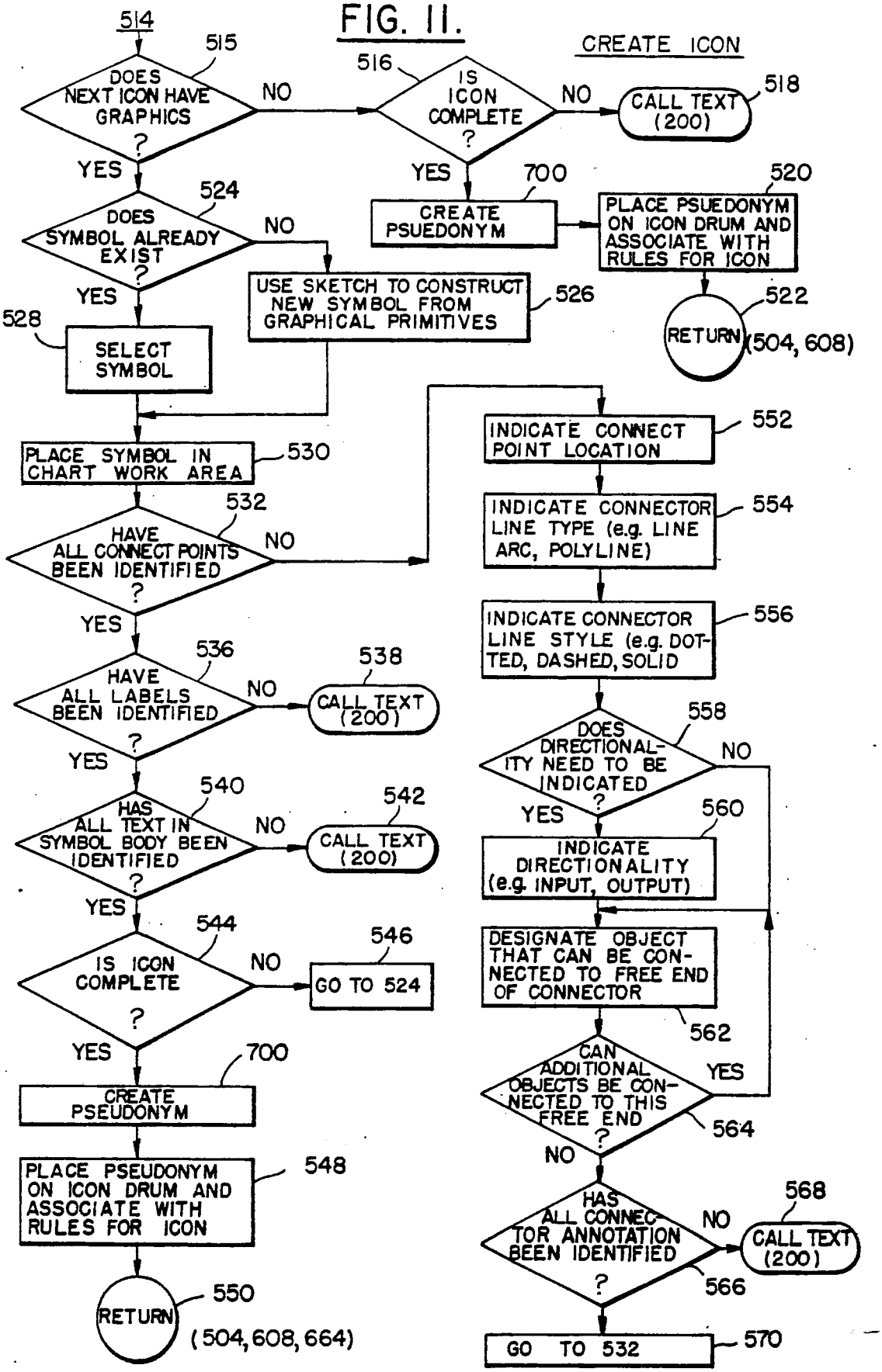
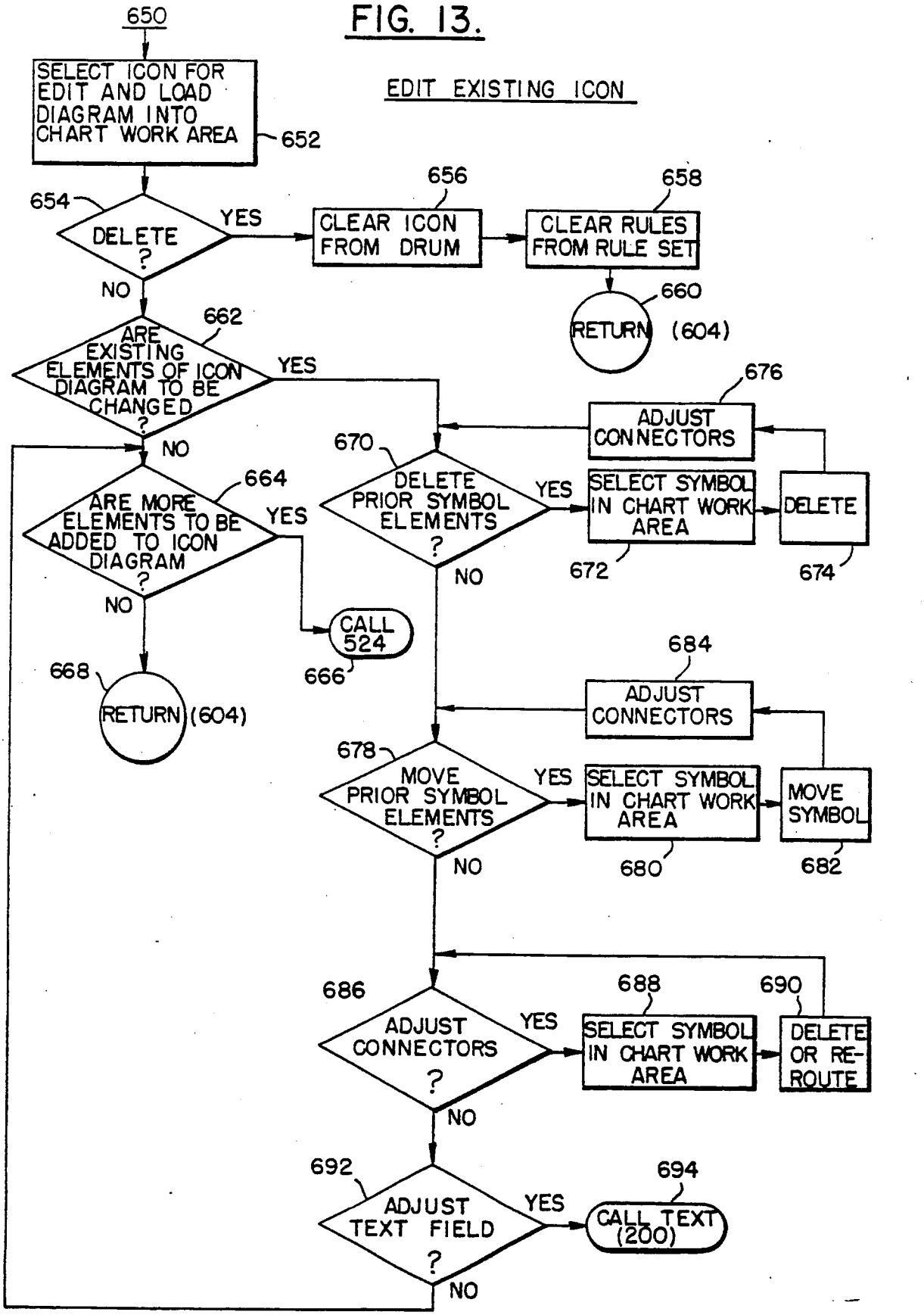

MODIFY EXISTING ICON DRUM
600
FURNISH NAME OF ICON DRUM TO BE MODIFIED
602
604
IS EXISTING ICON TO BE EDITED ?
YES
606
CALL 650
NO
608
IS NEW ICON TO BE ADDED ?
YES
610
CALL 514
NO
612
IS ICON DRUM COMPLETE ?
NO
YES
614
STORE MAP OF DRUM AS GRAPHIC FILE
616
LINK SET OF RULES TO BE ASSOCIATED WITH EACH ICON
618
ADD ICON DRUM NAME TO ANY ADDITIONAL FUNCTION DRUM DIRECTORIES AS NECESSARY
620
RETURN (110)

FIG. 11.



DEF017465

FIG. 13.



DEF017466

FIG. 14.

PSEUDONYM-SUBROUTINE

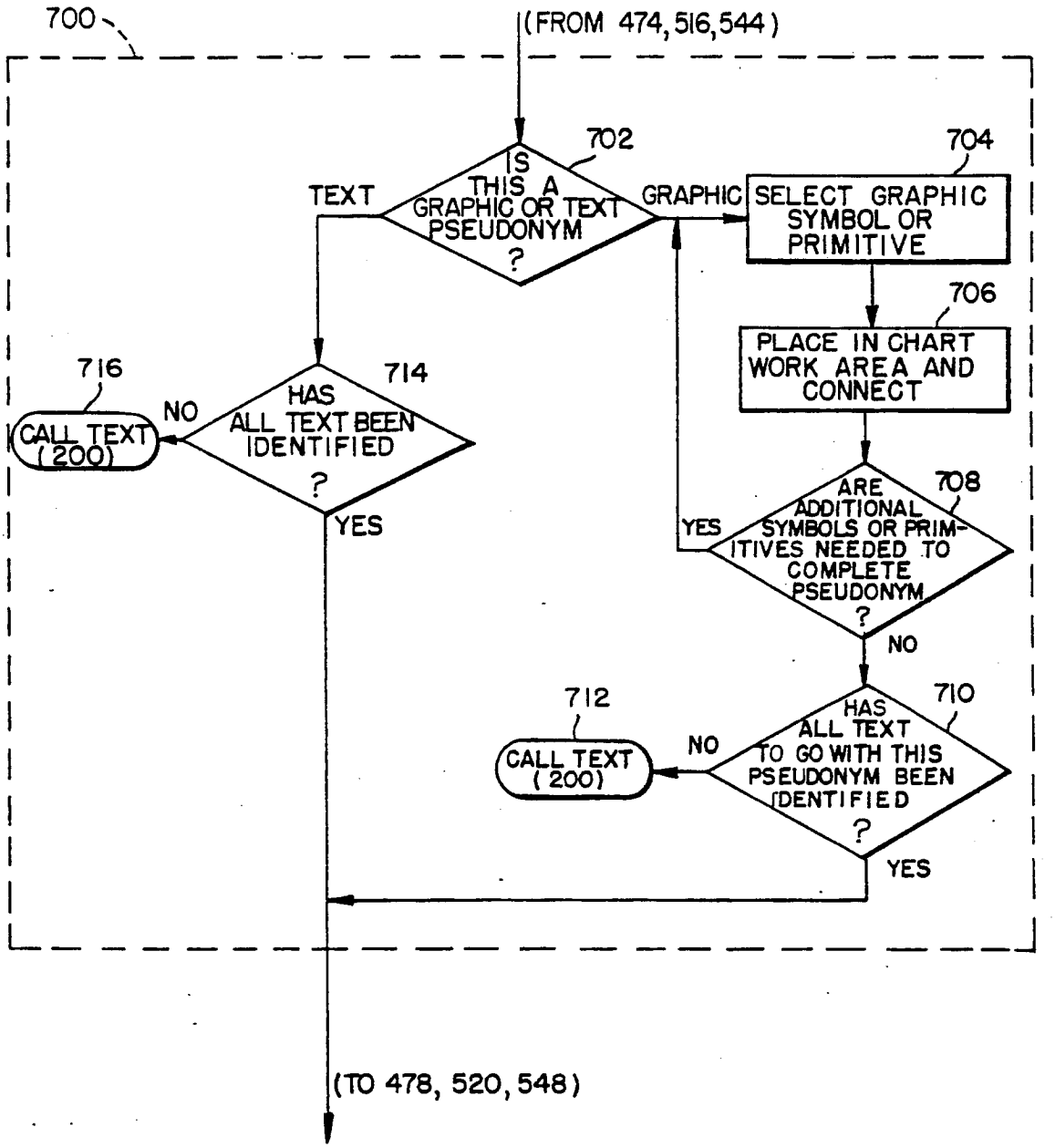
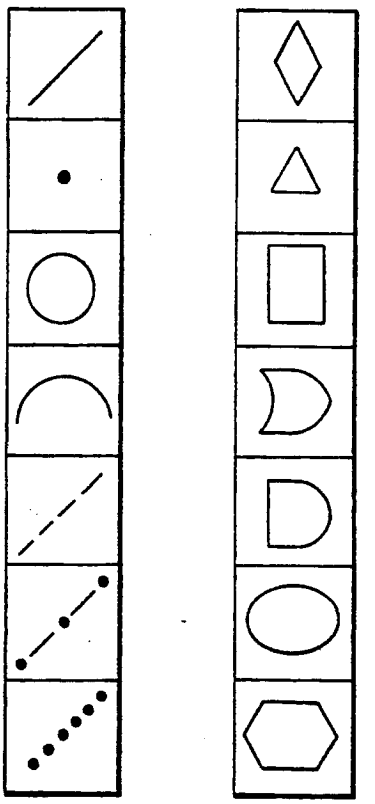


FIG. 15.

ADD	DELETE	MOVE	COPY	FIT	SIZE	REROUTE

FIG. 16.



SCHEMATIC DIAGRAM GENERATING SYSTEM USING LIBRARY OF GENERAL PURPOSE INTERACTIVELY SELECTABLE GRAPHIC PRIMITIVES TO CREATE SPECIAL APPLICATIONS ICONS

The present application includes a microfiche appendix.

BACKGROUND OF THE INVENTION

The present invention relates to computer systems and more particularly to a new type of interactive computer system for enabling a problem solving user to create (i.e. plan and evaluate) solutions to problems by building schematic diagrams representative of the solutions. The system of the present invention is useful in designing solutions which lend themselves to symbolic representation, e.g. in the form of schematic diagrams such as flow charts, process diagrams, and the like. Examples of designs which utilize such solutions are the layout of assembly lines for product manufacturing, the design of computer systems, computer software design, and the design of refineries and chemical plants. The interactive computer system disclosed herein also has application to the creation and use of "smart forms", e.g. for income tax reporting purposes, which are capable of satisfying the unique requirements of any problem solving user.

Past attempts to develop computer systems which are capable of creating problem solutions have been implemented through artificial intelligence techniques. Artificial intelligence is the branch of computer science that attempts to make machines emulate intelligent behavior. There has been success in enabling a computer to reason from knowledge in a limited domain, and in some instances computer programs implementing artificial intelligence techniques can exceed human performance. Such programs use a collection of facts, rules of thumb, and other knowledge about a given field, coupled with methods of applying those rules, to make inferences. These programs have been applied to such specialized fields as medical diagnosis, mineral exploration, and oil-well log interpretation. Since such programs often must make conclusions based on incomplete or uncertain information, they differ substantially from conventional computer programs which solve problems in accordance with pre-defined algorithms and complete data sets. The power of such systems results from entering large amounts of knowledge into the computer. It is such knowledge data, and not sophisticated reasoning techniques, that is responsible for the success of such "expert" systems. An introduction to such systems is provided in the article entitled "Expert Systems: Limited But Powerful", by William B. Gervarter, *IEEE Spectrum*, August, 1983, pgs. 39–45.

One problem with knowledge based expert systems is the vast amount of data which must be entered into the computer in order to provide a useable system. The more knowledge a system is given, the better will be its solution. There is a tradeoff, however, because greater search time is required when more information is entered into the system. Other drawbacks to such systems are their narrow domain of expertise, the requirement that problem solving users describe problems in a strictly defined formal language, and the extensive training required to teach problem solving users to use such systems.

The present invention provides a general purpose machine for designing problem solutions which can be represented in schematic diagram form, and avoids many of the problems associated with prior knowledge based systems. Although the system can be used to build problem solutions in any field in which solutions can be represented in schematic form, the system will be described herein primarly with reference to the automation of the proces used by programmers, data processing analysts, and system methodology designers to create computer software for commercial applications. The description of the invention in connection with the creation of computer software is not intended to limit the scope of the invention in any manner. Those skilled in the art will appreciate that as a general purpose system, the apparatus and techniques of the present invention will have broad application to the design of problem solutions in any field where such solutions can be represented in schematic diagram form.

Current methods of software design are time consuming and unsuitable for collaboration between many people. It would be advantageous to provide tools for software creation professionals which allow them to employ techniques derived from the world of computer aided design and engineering. These techniques would be utilized to create the logical analysis of the system whose design is being sought, to plan the solution itself, to identify the parts of the solution that can be implemented as independent modules, to identify the elements of communication between such independent modules in the solution's implementation, to design the independent solution components, and to test for their logical correctness prior to actually including the design solutions in a computer program. It would also be advantageous to be able to utilize functional modules previously designed in the context of other solutions for other problems, in the solution effort for the problem at hand. Such re-utilization of previously developed modules would increase the efficiency of the design process. The present invention relates to such a system.

In the system of the present invention, problem solutions can be generated in schematic diagram form in accordance with certain methodologies. The solutions are such that information can be extracted from a design at a graphical level and be used as the basis for subsequent input to automatic code generation tools or other automated functions, such as the generation of masks for fabricating integrated circuits. The system enables problem solving users to initially sketch what they conceive to be their current notion of a possible solution strategy. This mode of system use is referred to as the "chart" mode. After the sketching process has led to some degree of satisfaction and acceptance of the solution strategy, a formal rule-based schematic of the solution can be created which logically conveys each aspect of the solution prior to its implementation. The creation of such schematics occurs at the "schematic" level of the system. The formal schematic is then reviewed by the system to verify that it is correct with regard to all of the internal formal rules, details and mechanisms of the methodology that is employed to create it.

A key function of the present invention is the provision of means for enabling a methodology designer to create a library of logically or methodologically based schematic symbols, or graphic icons, and related formal functions which govern the employment and manipulation of the graphic icons when combined into a problem solution. The icons and functions ar created by the

methodology designer on an interactive basis with the system. Thus, during the creation of such icons and functions, the system prompts the methodology designer to identify, by way of example, the parameters and use of each icon or function. On the basis of these parameters, the system generates and stores a specific set of rules for each and every icon and function which completely and logically establish how each icon and function can be used to build problem solutions. In prompting a methodology designer to identify such parameters, the system requires the methodology designer to provide examples as to how the icon or function being created can be connected to other icons and/or functions. Such examples enable the system to generate the specific rules for the use of the icons and functions. Thus, the rules are built "by way of example".

As noted, the methodology designer can create new graphic icons and new formal functions. The new functions can be represented either in a text format or symbolically as "function icons". Such new functions can be built in three ways, to be explained hereinafter in detail. In general, these three methods comprise (1) concatenating function primitives already stored in the system; (2) concatenating function primitives with other existing complex functions; and (3) creating value tables for a function icon by assigning input and output values to all of the open ended connectors present on the function icon, which values are used by the system to establish a transfer function across the icon. The types of values which can be assigned when creating value tables are actual numerical values, indirect values (i.e., pointers to actual values), and inputs or outputs of other functions.

Unlike prior knowledge based systems, which require the entry and storage of a vast amount of information, the system of the present invention builds rules on the basis of procedure within a field of intent. Each such field of intent may be represented by its own library of icons and functions within the system. The system thereby precludes interference between intents. Since a methodology designer can create unique sets of icons and functions to accomplish desired intents from axiomatic primitives stored in the system, a truly general purpose machine results. Once a methodology designer creates a special library of icons and functions for a desired application (i.e., intent), the system can then accommodate any problem solving user who desires to build problem solutions for that application in schematic diagram form. The methodology designer has complete control over what applications the system can be used for and in the definition of the methodology to be used in building solutions for those applications. This is a substantial advance over prior systems, which can be used only for the creation of problem solutions according to an established methodology in the specific field in which the machine is "knowledgable".

SUMMARY OF THE INVENTION

In accordance with the present invention, an interactive rule based system is provided for generating problem solutions in schematic diagram form. The system operates and can be interfaced with on several levels. A first level of operation is engaged in by a "problem solving user", who builds schematic diagrams for applications available through existing machine functions. The first level problem solving user has an inventory of such functions, arranged in libraries or sets on a function drum, and an inventory of icons, arranged in libraries or sets on an icon drum and that together provide the building blocks necessary to use the system for an intended application.

A second, higher order level of operation is engaged in by a "methodology designer" who actually creates the libraries of functions and icons that are available to the problem solving user. It will be understood that the methodology designer and problem solving user can be the same or different persons; the role of each, however, is quite distinct. The methodology designer, through interactive use of the system, actually establishes the methodologies that govern the building of schematic diagrams by the problem solving user. The present invention is primarily concerned with this second level of operation, and in particular with the unique tools provided for use by the methodology designer.

The system includes a computer processor, a graphics terminal coupled to the processor, and means for providing a multi-portion split display on the graphics terminal. A plurality of functions and graphic and functional primitives are stored in the computer processor. Means are provided for enabling a methodology designer to employ the functions and select and arrange the graphic primitives using the graphics terminal to create a library of icons. Means operatively associated with the computer processor are provided for prompting a methodology designer to identify, by way of example, parameters which define each icon and its use. The system generates and stores a specific set of formal rules pertaining to the nature and use of each icon on the basis of the parameters identified. The stored formal rules are cross-referenced to the icon to which they pertain, so that whenever the icon is selected by a problem solving user for use in building a problem solution, the formal rules pertaining thereto will be accessed and applied.

In order to provide a friendly and consistent interface for a problem solving user, means are provided for symbolically displaying, in one portion of the split display, an icon drum comprising a set of icons from the icon library created by the methodology designer. Each such set of icons relates specifically to a category of formal schematic diagrams. Similarly, a set of functions is displayed on a function drum in another portion of the split display. Each such set of functions relates specifically to a category of operations applicable to the category of schematic diagrams which can be created from the icons displayed on the icon drum. Means are provided for enabling a problem solving user to access and select icons and functions displayed on the icon and function drums, and to build a solution to a problem by functionally arranging and coupling icons together on a chart work area portion of the split display. The system insures that the functional arrangement and coupling of icons is made strictly in accordance with the formal rules that apply to each of the icons so arranged and coupled and the functions which are employed to do so.

The system of the present invention can further comprise means for enabling a methodology designer to select and concatenate functions to each other and to icons, using the graphics terminal, to create more complex functions for display on the function drum. The complex functions can subsequently be used by a problem solving user to build problem solutions in the corresponding area of intent (i.e., for the corresponding application). Further, in order to create new functions, the system will analyze any new function icons created by a methodology designer to identify open ended connec-

tors, and prompt the methodology designer to assign input or output values to the open ended connectors that become part of the employment rules for the later use of the function icon analyzed. The input and output values assigned by the methodology designer establish a transfer function across the function icon. The function icon and values, in combination, form a new function for display on the function drum and for subsequent use in building problem solutions.

In prompting a methodology designer to identify, by way of example, the defining parameters for the intended use of each graphic or function icon, the system requires the methodology designer to define the points of connection to each icon. Further, the methodology designer is required to indicate, for each connection point, whether the point is an input, an output, or a bidirectional port. A methodology designer is also required to indicate the connector line styles (e.g., dotted, dashed, solid line, etc.) which are permitted to be connected to each connection point, and to indicate the connector line types (e.g., line, arc, polyline, etc.) which are permitted to be connected to each point. In addition, the methodology designer is required to indicate, through system prompts, what other objects (e.g. icons or functions) are permitted to be connected, through a connector, to each connection point. Other system prompts require a methodology designer to identify whether any annotation is to be associated with a connector, and if so, what type of annotation is to be permitted. In order to complete the formal rules which define the proper use of each icon, the methodology designer is required to identify any fixed and variable text and other forms of annotation which are to be associated with each icon.

By establishing a list of formal rules for the use of each icon and function, the system is able to enforce the rules during the building of problem solutions. Thus, when a problem solving user functionally arranges and couples different icons together at the first level of system operation to create a problem solution, the system continuously verifies that all of the formal rules for the particular icons being used and functions applied are complied with.

Any input means well known in the art can be used to enable a problem solving user or a methodology designer to interface with the system. In a preferred embodiment, an input device known as a "mouse" is used to enable the selection of functions and icons and their placement on the chart work area of the graphics terminal, and a keyboard is used to enable text and numbers to be input.

The system of the present invention is an intentional system. It is a procedural rule based expert system, and not knowledge based with inference rules as are most prior artificial intelligence expert systems. By creating new icons and functions, and inputting the parameters which establish the formal rules for each icon and function, a methodology designer can customize the system for any intended purpose. Thus, a methodology designer could develop icons and functions to enable the system to be used by a problem solving user to build flow charts for computer software creation, to create process flow diagrams for chemical engineering problems, to create schematic diagrams for electrical engineering problems, or to create any other type of chart which can be used to build solutions to specific problems.

The formal rules established for the system are axiomatic and, as such, the system is closed, complete, and consistent. Such formal rules are never modified or broken to fill a special need; once established there can be no exceptions to their application when the associated icons or functions are used by a problem solving user. Thus, those skilled in the art will recognize that the system of the present invention is significantly different from any computer implemented problem solving or design system known heretofore. In particular, such prior art systems operate in accordance with heuristic techniques (i.e. analogical, allegorical, metaphorical, and paradigmatic) and not the formal logical techniques (i.e. procedural, inductive, abductive, and deductive) applied in the present system. Heuristic based systems utilize rules of thumb or empirical knowledge to guide a problem solution and are judgmental in nature. Thus, they often result in ad hoc solutions which are not entirely consistent with past results. In the present system, the establishment of and adherence to formal rules insures consistency and provides a truly general purpose machine with which a methodology designer can implement any desired intent.

Since the system enables the creation of icons and functions which are each associated with specific formal rules within an intent, and the rules are always consistently enforced at all levels of system operation, problem solutions generated by the system can easily be supported and analyzed at various levels. For example, program description information can be extracted from a completed solution and be entered into a program description file for subsequent use in executing the problem solution. Thus, in the case of software creation, the extracted program information could be used by automatic code generators to produce actual software code. In the case where masks for integrated circuits are to be created, the program description information can be used to drive mask generators.

Information which can be extracted from problem solutions generated by the system of the present invention includes data of several kinds. For example, if the problem solution generated concerns the creation of computer software, control information will be extracted which governs the operating environment required for successful execution of the program when it has been converted into executable code. Also, the formal procedural description of the logical functioning of the program will be extracted, along with the formal description of the data types of the information employed within the program. If a problem solution concerns data flows rather than program flows, equivalent extractions could be made for use in a data description file as opposed to a program description file. The extraction of data from data files for future use is a process well known to those skilled in the art. In the present instance, extraction of information can be used to support requirements analysis of problem solutions to be generated, for decision support functions, to provide functional analysis of a solution which has been created, to provide classification and reclassification of extracted properties, to enable relational analysis, and to explore new applications (new rules) for the system. While the present invention is not concerned with any of these applications per se, it is important to recognize that the logical techniques used by the present system make the future provision of such applications obtainable in a straightforward manner.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a perspective view of the components of the system of the present invention;

FIG. 2 is an example display which might appear on the graphics terminal;

FIG. 3 is a flow chart showing the operation of the "Rule Tool" which enables a methodology designer to create and modify function and icon drums containing functions and icons having rules associated therewith;

FIG. 4 is a flow chart of the Rule Tool text subroutine;

FIG. 5 is a flow chart of the procedure for creating a function drum with the Rule Tool;

FIG. 6 is a flow chart of the procedure for modifying an existing function drum with the Rule Tool;

FIG. 7 is a flow chart of the procedure for editing existing functions with the Rule Tool;

FIG. 8 is a flow chart of the procedure for adding a new function with the Rule Tool;

FIG. 9 is a flow chart of the procedure for prompting open ended connectors;

FIG. 10 is a flow chart of the procedure for creating an icon drum;

FIG. 11 is a flow chart of the procedure for creating a new icon;

FIG. 12 is a flow chart of the procedure for modifying an existing icon drum;

FIG. 13 is a flow chart of the procedure for editing an existing icon;

FIG. 14 is a flow chart of the procedure for creating pseudonyms for functions or icons;

FIG. 15 is a graphical example of function primitives which are stored in the system of the present invention; and

FIG. 16 is a graphical example of graphics primitives which are stored in the system of the present invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

Turning now to FIG. 1, the system 10 of the present invention comprises a graphics terminal 12 with a display screen 14. Graphics terminal 12 is coupled to a computer processor 26, which also supports keyboard 16 and mouse 18. The graphics terminal, display screen and computer processor can be a personal computer ("PC") of a type well known in the art, such as the PC manufactured by Wang Laboratories of Lowell, Mass. Mouse 18 includes three switches 20, 22, and 24 which are assigned, for right-handed use respectively, the actions of select, cancel and evaluate. For left handed use, the position of the select and evaluate buttons can be reversed, to provide for corresponding finger use to depress a button.

The select button 20 indicates to the system, when pressed, the selection of a free space point, object, icon, or function based on the position of a cursor 29 controlled by mouse 18 on screen 14. To select a free space point, the cursor is positioned to the desired location on screen 14 and the select button 20 is actuated. A free space point is any point not occupied by, or within the tolerence of, an object. A marker is displayed at the selected point. An object is selected by positioning the cursor on or near (with a search tolerance) the object on screen 14 and actuating select button 20. An icon is selected by placing the cursor within a defined boundary box for the icon. A function is selected by simply moving mouse 18 so that the cursor on screen 14 is within a box enclosing the function, and then actuating select button 20.

Function selection can also be associated with evaluate button 24. To perform the association, mouse 18 is moved so that cursor 29 is within the desired function box, and then evaluate button 24 is actuated. The function selected will become associated with evaluate button 24 and, from that point on, actuating the evaluate button will produce the same result as positioning the cursor over the function and actuating the select button. The advantage of the association of a function with evaluate button 24 is the minimization of eye & hand movement.

Cancel button 22, when actuated, causes the system to disregard the last action. This is useful in overcoming an unintentional actuation of one of buttons 20 or 24, or to recover when a problem solving user or methodology designer has changed his or her mind with respect to the next action to be taken.

Keyboard 16 is provided in order to enable the input of numbers and text into the system. The design and use of keyboards and mouse devices and their interface with computer systems, including graphics terminals, executive work stations, computer aided design systems and the like is well known, and those skilled in the art are aware of how to implement such input devices. Any of the known systems for interfacing mouse 18 and keyboard 16 with computer processor 26 and graphics terminal 12 can be used in connection with the present invention. It is noted that alternative input means, for instance a graphics tablet, can be used instead of mouse 18.

FIG. 2 shows a sample multi-portion split display which can be output on screen 14 of graphics terminal 12 in accordance with the present invention. The display is "split" into system defined areas called regions. A region is defined to be a rectangular area of the screen enclosed by a border of lines. Each region serves a functional purpose, and is adjacent to another region. None of the regions overlap.

In FIG. 2, region 28 is the chart work area, and is a view of the drawing or a portion thereof that is presently being worked on by a problem solving user or methodology designer. Chart work area 28 may be thought of as a window through which one is viewing a chart that sits beneath. It is within this region that schematic diagrams are created and edited by a problem solving user, and new icons and functions are created by a methodology designer.

The entire available area onto which a chart may be created is called the drawing space. At any one time, the drawing region in chart work area 28 is a display of a part, or all, of that drawing space. Since only a portion may be visible, means is provided to view the other areas of the drawing space. Such means is provided by navigation region 58.

Navigation region 58 contains a rectangle 59 called the view port. The view port represents that portion of the chart currently visible in chart work area 28. For example, if view port 59 is half the size of the navigation region 58, and directly in the center, then exactly half of the drawing space is visible in chart work area 28. Since view port 59 is centered, the half that is visible in chart work area 28 is the center of the drawing space.

To view a different portion of the drawing space, a function which is built into the system, called the move function, is used to move the view port 59 to the desired location within navigation region 58. In order to scale

the size of the drawing space, the size of view port **59** must be altered. This is done by using a size function built into the system. Once the move and/or size functions are completed, the drawing region is refreshed to display the correct portion of the chart at the correct scale.

Another region on display screen **14** is known as the icon drum region **60**, which displays a portion of what can be thought of as a band of icons upon a drum. Icons represent various symbols, lines, and text that can be selected for placement in the drawing region of chart work area **28**. The band of icons that is placed upon the drum in icon drum region **60** depends upon the level in which one is working on the system and the type of schematic diagram that is to be created or edited by a problem solving user, or the new function or icon to be created by a methodology designer. There are actually two drums within region **60** each having their own band of icons and which may be individually manipulated. One band contains icons **62, 64, 66, 68, 70** and **72** (and others not visible). The other band contains icons **74, 76, 78, 80, 82,** and **84** (and others not visible). If a desired icon is not displayed in region **60** when needed, the drums can be scrolled until the desired icon appears.

Underneath each icon drum are two pair of arrows **50, 52** and **54, 56**. These are scroll control arrows and represent the function of spinning the drum above the arrows selected. If, for example, arrow **50** is selected, the drum containing icons **62–72** will be spun upward. Selecting arrow **52** would spin this icon drum downward. Similarly, selecting either of arrows **54** or **56** would spin the icon drum comprising icons **74–84** upward or downward, respectively. The scroll control arrows are contained in region **44** of display screen **14**. Also contained in region **44** are scroll arrows **46** and **48** which are used to horizontally scroll the function drum **40** as described below.

Region **40** in display screen **14** is the function drum region and contains the functions of the system that are applicable to icons or schematic diagrams or their parts. As shown in FIG. **2**, the functions in region **40** are in the form of text strings. As will be described hereinbelow, however, region **40** can also contain symbols which represent functions or function primitives. If a desired system function is not displayed in region **40** when needed, right and left scrolling arrows **46** and **48** can be used to spin the function drum toward the right or toward the left until the desired function appears.

Region **42** on display screen **14** is a prompt region which serves two purposes. First, the system uses region **42** to display prompts, which are usually in the form of text strings, to the problem solving user or methodology designer. Second, any textual data entered by a problem solving user or methodology designer through keyboard **16** is echoed in prompt region **42**.

An informational region **31** on display screen **14** provides a problem solving user or methodology designer with four selectable functions presented by text strings. These functions are labeled "parameters", "clear", "exit", and "help". The selection of the parameters function **32** brings up all of the parameters which can be set by a problem solving user or a methodology designer within the system. Examples of such parameters are text height, left or right handedness, line spacing, a parameter known as drum roll which defines how many icons are scrolled when the scroll arrows are used, and the scaling factor for defining the percentage any object is scaled when the size function is applied.

The clear function **34** in informational region **31** is equivalent to multiple cancels. If the problem solving user's or methodology designer's intent is to back up to the point of the previously committed function, the cancel function could be selected as many times as needed, or the clear function could be invoked once.

Help function **38** in informational region **31** displays information about objects or functions selected by a problem solving user or methodology designer. The information appears in a page which overlaps the drawing region in chart work area **28**. Those skilled in the art will recognize that various types of information can be provided to a problem solving user or methodology designer by accessing the help function.

Exit function **36** in informational region **31** enables a problem solving user or methodology designer to exit from his current work with the system, saves the work and the status of the session and returns the display back to an alphanumeric menu for selecting the next activity to be accomplished with the system.

Another region of display screen **14** is error region **30**. This region is used by the system to display error messages. The message displayed describes an invalid action taken by the problem solving user or methodology designer. The message will remain visible until the next action is taken, i.e., either one of buttons **20, 22,** or **24** is actuated on mouse **18**, or a keystroke is made on keyboard **16**.

Before discussing the heart of the present invention, which is the provision of means for enabling a methodology designer to create new icons and functions with a set of rules pertaining to each new icon and function, it is necessary to explain how the basic system can be used to sketch, in graphic form, on chart work area **28** of display screen **14**. This basic level of system operation is referred to as the "sketch" mode.

The sketch mode provides a free form diagramming capability with no restrictions imposed by the system. A variety of creation and editing functions exist. The problem solving user interfaces with the system to create sketches through mouse **18**. The relative movement of mouse **18** causes a corresponding movement of a cursor **29** on chart work area **28** of display screen **14**. A problem solving user may continually move the cursor near an icon on the icon drum **60** or a function on the function drum **40** and actuate select button **20**. This will associate mouse **18** with the object (icon or function) selected. If, for example, cursor **29** is moved into the region of icon **74**, and select button **20** is actuated, a rectangle (the object of icon **74**) will become associated with mouse **18**. Mouse **18** can then be moved to drag the rectangle across chart work area **28** until a desired location is reached. Upon activation of the select button, a rectangle will be deposited at the desired location in chart work area **28**. When a function is to be used, mouse **18** is moved to place cursor **29** over the desired function in function drum **40**, and the select or evaluate button is actuated to associate the function with mouse **18**. By manipulating functions and icons with mouse **18**, and placing icons on chart work area **28** at desired locations and with desired connections, schematic diagrams can be drawn. Interaction with other regions of display screen **14** is achieved through the use of mouse **18** in a similar manner. It is noted that in the system of the preferred embodiment, data may be selected in the postfix sequence, i.e. where the function selection occurs

after data selection, in the prefix mode, where function selection preceeds data selection, or in the infix mode, where function and data selection occur alternately.

Various classes of icons may appear in the icon drum 60. The majority of icons are representations of symbols, which have meaning with respect to a particular type of chart the problem solving user is building. The creation of this type of icon by a methodology designer in accordance with the present invention is discussed later on. It is noted that geometry represented by an icon may or may not be identical to the geometry actually shown in the icon drum. Thus, where the actual symbol is too complex to fit within the icon drum display portion of display screen **14**, a more simplified symbol (pseudonym for the actual symbol) will be shown. However, when the icon is placed in a chart, the symbol's actual geometry will appear. Such symbols may contain geometry consisting of lines, arcs, circles, other symbols and other geometric or graphical primitives or their components, as well as text fields.

In its simplest mode of operation, the present system can be used to create a "throw-away", electronic paper form of a diagram which a problem solving user can use for discussion or initial concept purposes. This mode of operation is referred to herein as the "chart mode". Once a problem solving user decides on a solution strategy, a schematic diagram mode can be entered which will enable a problem solving user to generate an actual problem solution in terms of a formal schematic diagram.

The icon drum initially supplied with the system will contain icons for use during the chart mode and during the schematic diagram mode of operation. The icons suppled for the chart mode are those for each type of line, arc, polyline and other graphical primitives as may be supplied. These icons may be depicted by solid, dashed and potentially other line styles.

Lines are scalar objects with no direction. They have no pre-defined association with other graphic primitives, icons, diagrams, or their parts. Lines may be attached by placing the end point of the line within the tolerence of the geometry of any graphic primitive, icon, diagram, or part of a diagram.

A polyline is a multisegmented line. Polylines are considered to be one object for the purposes of the system's functions applicable to icons and diagrams or their parts.

An arc is defined by three points; namely, the first end point, a point along the arc, and the other end point, in that order.

Additional primitive graphic icons are provided for use when building schematic diagrams. These include connectors, polyectors (poly-connectors), and conarcs.

Connectors are a special class of line or arc that always attach to a symbol at a connection point. Connectors may have text fields associated with them much like symbols do. They may have direction or they may be scalar. A connector is a single line or a single arc of any style (e.g. dashed, dotted, solid line, etc.). If it is desired to attach a connector to a symbol, the connector element must attach to a connection point on the symbol, even though the selected point was only near the actual connect point. That is, there are predefined points on a symbol which may be connected to by a connector element. Thus, connectors are distinct from free form lines which may attach to a symbol's geometry for stylistic effects of annotation, but which have no logical connection denoting a communication path between two other elements.

A polyector is a multi-segmented connector to allow the routing of a connection from one object to another where a single straight connector path would interfere with other geometry. A polyector is considered to be one object for the purposes of the system's functions applicable to icons and diagrams and their parts. A polyector is also considered to be one object in establishing logical connectivity between symbols.

A conarc is a connector arc. It is defined in the same way as a regular arc, but it connects to either the ends of connectors, polyectors, or connection points on symbols. Conarcs may also have text fields associated with them and they may have a direction.

The icon drum in the most basic form of the system of the present invention contains three text icons. One such icon is for left justified text, one for center justified text, and one for right justified text. Other, less frequently changed parameters for text, such as font style, vertical justification, and height, can be set using the parameter table function. The text icons provide the functions of adding free form text to the chart. Other forms of text, such as a "label", are specific to a symbol or connector. Such other text forms will be discussed below in connection with the formation of new icons and functions.

Regardless of the type of text or its association or lack thereof, it is always entered in the same fashion. Assuming that all text parameters are correctly defined, an origin point is selected by moving cursor **29** to a desired position on chart work area **28**. This is the point used as the basis for the text justification. When the text function is selected, a text cursor appears in the prompt region **42** of display screen **14**. Simultaneously, a very narrow rectangle appears in the drawing region at the origin point. As text characters are input from the keyboard, they appear in the prompt region in the machine font. Simultaneously, the rectangle in the drawing region expands horizontally to show the area the text string would cover. Depending on the justification, the rectangle would expand to the right for left justification, to the left for right justification, and equally in both directions for center justification.

Each time a terminator (e.g. carriage return or other multiple control key sequence) is entered from the keyboard, the rectangle disappears, the text font replaces it in the chart, the prompt region is cleared and the text cursor reappears at the left most character position of the prompt region awaiting another string. The rectangle then reappears at the next indicated line on which text is to be placed in the drawing region. If no more text is to be input, either another function is selected or a null line is entered. If more text than can fit in the prompt region is required on one line, the input string will scroll horizontally in the prompt region.

As noted above, the system of the present invention is operable in a chart mode and a schematic diagram mode. The invention is concerned with the schematic diagram mode, and more particularly with the provision of tools for use by a methodology designer to enable the creation of new icons and functions for use in building schematic diagrams. The icons created by the methodology designer using the tools are formal symbols within an adopted schematic diagram methodology (e.g. for the development of computer software), rather than being general purpose graphic tokens such as those used in the chart mode. These formal icons are used for

the creation of schematic drawings and enable the system to accomplish interactive design rule checking within the adopted methodology. All of the formal rules and procedures that are attributed to any given step of the methodology, and which can be further attributed to a graphic symbol or token of some form, are associated with the formal icons on the icon drum of the system. In this manner, as a problem solving user selects an icon for use in placing an instance of the represented symbol in the schematic diagram of the solution, the system concurrently, and at a very high interactive rate, continuously checks to insure that the formal design rules associated with the instance of that symbol are being adhered to by the problem solving user. Thus, even though the data content (as furnished by the problem-solving user) associated with the symbol may not be exactly correct, the logical structure of the schematic as a methodological description must be precisely accurate. The schematic diagram created at this level of system use is not a "throw-away" diagram. Rather, such schematic diagrams are problem solutions which may be used, revised, analyzed, or transformed by other, higher level systems. Such opportunities are available as a result of the methodologically accurate, complete, and logically consistent schematics which are generated by the system of the present invention.

The unique and novel features of the present system will now be explained in detail, with reference to FIGS. **3** through **14** of the drawings. FIGS. **3** through **14** inclusive are flow charts which set forth the operation of system software referred to as the "rule tool". The rule tool allows a methodology designer to create new icons and functions. The new icons and functions will then be accessible to a problem solving user for use in schematic diagram creation. The definable icons are symbols and connectors. Before describing how to create such icons, it is necessary to define what the icons are.

Symbols are the basis of the schematic diagramming process. They are the major part of the schematic diagrams, and are usually of the most interest. They can be of varying complexity. At the sketch mode of system operation, they consist of just the graphics for the symbol and the icon that represents the symbol on the icon drum. No rules of use for the symbol are associated therewith at the sketch level.

At the schematic drawing mode, a given symbol comprises not only graphics but also a specified number of connection points. A connection point defines where a connector may attach and the nature of that attachment. The direction of connection is specified as being input, output, or both (bidirectional). Each connection point also defines what other symbol(s) it may be connected with. A class of symbols or named diagrams may be specified for valid connections.

Symbols for the schematic drawing mode may also contain text fields. There are two major categories of text; namely, extractable and non-extractable. Higher level programs can use these categories to know which text fields to extract. Both types of text can be defined in terms of syntax rules for the text fields. Each text field has a position and may include text that always appears in every instance of the symbol and/or text that varies from instance to instance which the problem solving user must enter every time the symbol is placed. The placement of specific text could optionally have its own syntax rules of number of characters, or a list of valid entries. Such a text field could, for example, be a volume label which always has the form: "vol=[some character string]". The text field would define "vol=" to always appear, and have the problem solving user enter the desired volume value whenever the symbol is placed.

It is also possible to create a symbol which is composed of other symbols, and the same rules will apply. Such compound symbols look the same as a simple symbol, and contain a varying number of connection points with rules on the permissable type of connections. The only difference is that the connection points may be spread over many symbols. An example of the use of a compound symbol would be in diagramming subroutines. The chart for the subroutine could be diagrammed, and then turned into a compound symbol, with the input and output rules defined by the connection point rules of any unconnected points of the subroutine diagram. That subroutine symbol could then be used in other diagrams and the proper checking would be performed.

Connectors are used to connect symbols. They have a graphical description (e.g., linear single segment, linear multi-segment, or curvilinear), and a line style of either solid, dashed, or other form. They can have an icon representation for the icon drum. They can also have extractable and non-extractable text fields as described above for symbols.

Turning now to FIG. **3**, box **100** provides for a methodology designer's access to the rule tool. As indicated by box **102**, which is connected to the flow chart of FIG. **3** by a dashed line, the sketch program described above is used in combination with the rule tool to enable a methodology designer to create new formal icons and functions. Unlike the system operation by a problem solving user (e.g. during sketch) the use of the rule tool by a methodology designer results in the creation of icons and functions which each have a specific set of rules pertaining to the use thereof.

Once a methodology designer has accessed the rule tool, several prompts are displayed in the prompt region **42** of display screen **14** to determine what activity the methodology designer desires to undertake. At box **104**, the system determines if it is the intent of the methodology designer to create a new function drum. If this is the intent, then the routine entitled "create function drum" (set forth in FIG. **5**) is called at box **105**. Otherwise, the rule tool proceeds to box **106** which inquires whether the methodology designer wishes to modify an existing function drum. If so, the routine entitled "modify existing function drum" (set forth in FIG. **6**) is called at box **107**. Otherwise, the methodology designer is asked at box **108** whether a new icon drum is to be created. If a new icon drum is to be created, the routine entitled "create icon drum" (set forth in FIG. **10**) is called at box **109**. Otherwise, the methodology designer is prompted at box **110** to determine if modifications are to be made to an existing icon drum. If this is the case, the routine entitled "modifiy existing icon drum" (set forth in FIG. **12**) will be called at box **111**. Otherwise, control will pass to box **112** which asks the methodology designer if more rule tool work is to be done. If so, control passes back up to box **104** and the process repeats. Otherwise, the methodology designer exits the rule tool at box **114**.

When the rule tool calls another routine, such as indicated by boxes **105**, **107**, **109** and **111** of FIG. **1**, control immediately passes to the routine called, and returns back to the box prior to the "call" box shown in the flow chart when a corresponding "return" command is reached in the called routine. Thus, for exam-

ple, if a methodology designer intends to create a new function drum, as determined at box **104** in FIG. **3**, the "create function drum" routine will be called by box **105**. Once a return from the "create function drum" routine is reached (box **322** shown in FIG. **5**), the program flow will be returned to box **104** of FIG. **3**.

In the flow charts provided in FIGS. **3** through **14**, "return" boxes are annotated with the numbers of the boxes from which the routine could have been called, and to which control should be returned, as appropriate, once the routine has been executed. Those skilled in the art will appreciate that several returns may be nested, and control will be passed from a return box back to the box which called the routine in the proper nested order.

If a methodology designer indicates at box **104** that a new function drum is to created, the "create function drum" routine (designated by numeral **300** and shown in FIG. **5**) will be accessed. As shown in FIG. **5**, this routine requires the methodology designer, at box **302**, to label the new function drum with the name of the category of operations to which the new function drum will pertain. At box **304**, the methodology designer is required to enter the name of a chart type which can be used with this function drum. Such chart type might be, for example, flow charts for computer software design, flow charts for chemical processes, electrical circuit schematics, and the like. After entering the name of one chart type, the methodology designer is required at box **306** to indicate if there is another chart type which can be used with this function drum. If so, control is returned to box **304**, and the process repeats until all of the chart types which can be used with the function drum have been identified.

When there are no more chart types to be identified, control passes to box **308** where the rule tool generates a chart type menu for the function drum being created. The chart type menu lists all of the different chart types which can be used with the function drum. The menu is stored so that it can be displayed at a future time to a problem solving user who selects this particular function drum for use in building a schematic diagram. At such time, the chart type menu will be displayed to the problem solving user so that the desired chart type can be selected.

At box **310**, the rule tool associates the chart type menu with the name of the category of operations that the function drum was labeled with in box **302**. In box **312**, the name of the category of operations is added to the system operations menu. The system operations menu is displayed to a problem solving user at the beginning of the schematic diagram mode of system operation. The problem solving user selects a category of operations, thereby defining the applicable function drum. The chart type menu associated with function drum is then displayed and the problem solving user selectes a chart type as described above.

At box **314** of the "create function drum" routine, the methodology designer is prompted to indicate if the function drum being created is complete. If the answer is no (which will be the case the first time the methodology designer is asked the question), a routine entitled "add new function" will be called as indicated at box **316**. The "add new function" routine is shown in FIG. **8**.

New functions can be created in three different ways. A brand new function can be created by concatenating function primitives stored in the system. Examples of function primitives are the "AND", "OR", and "NOT" logic operators. Alternatively, a new composite function can be created by concatenating function primitives with other existing complex functions. "Macro-functions" can be created by concatenating a plurality of existing complex functions.

In addition to the above, there is another type of function which a methodology designer can build using the present system. These functions are, in effect, value tables for an icon or icon/function combination which has been built. Such a value table is created by identifying all connectors which exit from a given symbol, and assigning values (pointer values, derived valves, or literal values) to all non-terminated inputs and outputs. Straightforward, known techniques (e.g. calculus of variation) are used to establish a transfer function for the symbol in accordance with the assigned input and output values.

When the "add new function" routine shown in FIG. **8** is entered at **460**, box **462** requires the methodology designer to indicate whether a prior function or function primitive is to be added to the chart. If so, control passes to box **464** which requires the methodology designer to select the prior function or function primitive to be added to the chart. The selection of the prior function or function primitive will be made by using mouse **18** to select the appropriate symbol from the function drum **40** displayed on display screen **14**. The use of mouse **18** to select symbols from the function and icon drums has already been explained hereinabove.

Once the desired symbol has been selected, the methodology designer is required at box **466** to place it on the chart work area and connect it to other symbols on the chart as appropriate. Control then passes to box **468** which asks the methodology designer whether any connector annotation is necessary. If no such annotation is necessary, control reverts to box **462** and the procedure repeats until all necessary symbols have been selected, placed, and connected on the chart. In the event connector annotation is necessary, box **468** passes control to box **470** which calls a routine entitled "text".

The "text" subroutine is shown in FIG. **4**. After entering the text subroutine at **200**, the system asks the methodology designer at box **202** if there is text already present in the chart being worked on. If not, the methodology designer is required at box **205** to designate the text start point, and at box **207** to select the text justification. Otherwise, control passes from box **202** to box **204** which requires the methodology designer to find the start point of the existing text.

The rule tool next steps the methodology designer, one character position at a time, through the new text to be added (box **208**). Creation or alteration of text commences at the designated start point and the procedure is the same whether the methodology designer is creating a new text field or modifying an existing string of text. At box **208**, the methodology designer is required to decide whether the next character position is to be retained as is or altered. If it is to be retained, control passes to box **212** which merely spaces to the next character, and redirects control to box **208**. If the character is to be altered, control passes to box **214** where the methodology designer is required to indicate if the new character is to be fixed text or variable text. If fixed text is desired, control passes to box **216** which requires the methodology designer to enter the desired fixed character. Control then passes back to box **208**. If variable text is desired, control passes to box **218** which requires the

methodology designer to enter the variable text designator which indicates to the system that a problem solving user will ultimately have to enter text at this character position. Control is then passed back to box **208** and the process continues until the methodology designer indicates that there is no next character position (i.e., the end of the text line has been reached). At this point, control will pass to box **220** where a line termination designator is entered. At box **222** the methodology designer will be prompted to indicate if another line of text is desired. If so, a line advance is generated at box **224** and control is passed back to box **208** so that the new line can be created. When all of the lines of text have been created, control will be passed from box **222** to box **226** which returns control back to the point from which the text subroutine was entered. In the present example, the text subroutine was called by box **470** of the "add new function" routine shown in FIG. **8**. Thus, control will be passed back to box **468** shown in FIG. **8**, which is the box that precedes the call text box **470**.

Turning back to FIG. **8**, once all prior functions and function primitives necessary to create a new function have been selected, placed on the chart work area, and connected to the chart, and all connector annotation has been accomplished, control will be passed from box **462** to box **472**. At this point, the system analyzes the symbol currently on the chart work area to locate open ended connectors. When an open ended connector is found, box **474** passes control to box **476**, which prompts the methodology designer to enter the appropriate input or output value for the open ended connector.

Box **476** is shown in greater detail in FIG. **9**, entitled "prompting of open ended connectors". As shown in FIG. **9**, when an open connector is found the system records the connector and its type as indicated at box **480**. Then, control is passed to box **482** where the methodology designer is prompted to indicate if a literal, pointer, or derived value is to be associated with the open end of the connector. If a pointer value is designated, control passes to box **484** and the methodology designer is required to enter the pseudonym of the pointer. A pointer value is not a numerical value itself, but rather an address or "pointer" to a location which contains the value to be associated with the open ended connector. Once the pseudonym of the pointer has been designated, control passes from box **484** out of box **476** and on to box **472** of FIG. **8**.

If, at box **482** of FIG. **9**, the methodology designer indicates that a derived value is to be associated with the open ended connector, control is passed to box **486** which requires the entry of the pseudonym of the function from which the value is to be derived. From box **486**, control exits box **476** and continues on to box **472** of FIG. **8**.

If the methodology designer indicates at box **482** of FIG. **9** that a literal value is to be associated with the open end of the connector, control passes to box **488** where the methodology designer indicates whether a single value, range, or multi-element set of values is to be used. If a single value is indicated, control passes to box **490** where the methodology designer enters the value. Control is then passed out from box **476** and on to box **472** of FIG. **8**.

If a range of values is to be associated with the open ended connector, control passes from box **488** of FIG. **9** to box **495** where the methodology designer enters the lower bound of the range. Then, at box **496**, the methodology designer is prompted to designate if the lower bound indicated at box **495** is exclusive or inclusive. Control then passes to box **497** where the upper bound of the range is entered. Then, at box **498**, the upper bound is designated as being exclusive or inclusive. Once the range has been entered, control passes from box **498**, out from box **476**, and on to box **472** of FIG. **8**.

If, at box **488** of FIG. **9**, the methodology designer indicates that a set of values is to be associated with the open ended connector, control passes to box **492** and the next value in the set is entered. At box **494** the system prompts the methodology designer to indicate if there is another value in the set. If so, control is passed back to box **492** and the process continues until all values in the set have been entered. Once the set is complete, control passes from box **494**, out from box **476**, and on to box **472** of FIG. **8**.

After control is passed back to box **472** from box **476**, the search for open ended connectors continues until values have been assigned to all of the open ended connectors. Then, control is passed to box **700** for the creation of a pseudonym for the new function. The pseudonym is the symbol which will be stored on the function drum for future access of the new function created. Although the pseudonym can be the entire symbol built by the methodology designer through the use of the "add new function" routine, such will only be possible if the symbol is simple and will fit on the function drum. Otherwise, an abbreviated form (i.e., pseudonym) will have to be created to identify the new function on the function drum. The creation of pseudonyms at box **700** is set forth more completely in FIG. **14**, which shows box **700** in greater detail.

As shown in FIG. **14**, box **702** prompts the methodology designer to determine if a graphic or text pseudonym is being created. If a graphic pseudonym is desired, control passes to box **704** where the methodology designer is required to select the graphic symbol or primitive from which the new pseudonym is to be built. Box **706** requires the methodology designer to place the selected symbol or pseudonym in the chart work area and to connect it to any other symbols or primitives already in the chart work area. At box **708**, the methodology designer is required to specify if there are additional symbols or primitives needed to complete the pseudonym. If so, control returns to box **704** and the process continues until the symbol for the pseudonym is complete. Then, control is passed to box **710** where the methodology designer is required to indicate if any text which goes with the pseudonym symbol has been identified. If not, the text subroutine (numeral **200**, FIG. **4**) is called at box **712**. After all text has been identified, control passes from box **710** out from box **700** and onto the next successive box.

If, at box **702**, the methodology designer indicated that a text pseudonym was to be created, control passes to box **714** which determines if all text has been identified. If not, the text subroutine is called at box **716**. Once all text has been identified, control is passed from box **714** to the next box in the flow chart.

In the present example, the next succeeding box is box **478** in FIG. **8**. At this point, the pseudonym is placed on the function drum and associated with the value set applicable to the function. At box **480**, control is returned to the portion of the rule tool from which the "add new function" routine was originally called. In

the present example, control is returned to box **314** in FIG. **5**.

Once the function drum being created by the methodology designer is complete, control is passed from box **314** to box **318** and a map of the function drum is stored in the system as a graphics file. At box **320**, the set of rules to be associated with each function in the function drum is linked to the function drum for later recall any time the function drum is selected by a problem solving user. At box **322**, the rule tool returns to the point from which the "create function drum" routine was called which, in the present example, was box **104** of FIG. **3**.

If the methodology designer desires to modify an existing function drum, box **106** of FIG. **3** will pass control to box **107** which calls the "modify existing function drum" routine shown in FIG. **6**. From entry point **400**, this routine proceeds to box **402** and requires the methodology designer to furnish the name of the function drum to be modified. Box **404** then prompts the methodology designer to determine if an existing function on that drum is to be edited. If so, box **406** calls the "edit existing function" routine shown in FIG. **7**.

From entry point **420** in the "edit existing function" routine, control proceeds to box **424** which requires the methodology designer to select the function for edit and to load the diagram for that function in the chart work area. The diagram loaded into the chart work area can comprise function primitives, prior functions, or prior icons to be edited. Control then passes to box **422** which prompts the methodology designer to determine if the diagram loaded into the chart work area is to be merely deleted. If so, the diagram is deleted from the function drum at box **426**, and the rules associated therewith are cleared from the rule set at box **428**. Box **430** then returns to box **404** of FIG. **6** from which the "edit existing function" routine was called.

If the existing function is to be edited and not merely deleted, control is passed to box **432** which requires the methodology designer to indicate if there are specific prior functions or function primitives to be deleted from the diagram representing the function to be edited. If so, at box **433** the methodology designer is required to select the symbol of the prior function or function primitive in the chart work area to be deleted. This symbol is deleted at box **434**, and the methodology designer is required at box **435** to adjust the remaining connectors in the diagram as necessary. Control is then returned to box **432** where the process can repeat or continue on to box **436**. At box **436**, the methodology designer is required to indicate if there are prior functions or function primitives to be moved in the diagram currently in the chart work area. If so, the symbol to be moved is selected at box **437**, moved at box **438**, and necessary connector adjustments are made at box **439**. Again, the process repeats until all moves of prior functions or function primitives are completed.

Control next proceeds to box **440** which requires the methodology designer to indicate if any more connectors are to be adjusted. If so, a connector is selected in box **441**, deleted or rerouted in box **442**, and the process continues until all connectors have been adjusted. Control next passes to box **444** which determines if any text fields are to be adjusted. If so, the text subroutine is called at box **446**, the text fields are adjusted, and control is returned to box **444**. Once all text fields have been adjusted, the rule tool proceeds to enter the "add new function" routine shown in FIG. **8** and already discussed hereinabove. Upon completion of the add new function routine, box **480** (see FIG. **8**) returns control to box **404** of FIG. **6**.

When there are no more existing functions to be edited, control is passed from box **404** of the "modify existing drum" routine to box **408** thereof, and the methodology designer is prompted to indicate if a new function is to be added to the function drum. If so, the "add new function" routine shown in FIG. **8** is called at box **410**. Once the new function has been added, box **480** of the "add new function" routine returns control to box **408** of FIG. **6**.

When no additional new functions are to be added, box **412** prompts the methodology designer to indicate if the function drum being modified is now complete. If not, control is returned to box **404** and the process described above continues. Once the function drum is complete, control is passed to box **414** and the rule tool proceeds to box **318** of the "create function drum" routine shown in FIG. **5**. At box **318**, the map of the function drum just modified is stored as a graphics file. At box **320**, the set of rules to be associated with each function on the modified function drum is linked for later retrieval with each function when the function is selected by a problem solving user operating the system in the schematic diagram mode. At box **322**, control is returned back to box **106** in the rule tool flow chart of FIG. **3**.

After any modification of existing function drums has been completed, control passes from box **106** to box **108** of the rule tool flow chart, and if a new icon drum is to be created, box **109** calls the "create icon drum" routine shown in FIG. **10**. After entering this routine at **500**, control is passed to box **502** which requires the methodology designer to label the new icon drum being created with a name. The name will be indicative of the methodology and chart type to which the icon drum relates. At box **504**, the methodology designer is prompted to indicate if the icon drum being created is complete. The first time this box is reached, the drum will not be complete and control will be passed to box **505** which calls the "create icon" routine shown in FIG. **11**.

Upon entry to the "create icon" routine at **514**, control passes to box **515** which prompts the methodology designer to indicate whether the next icon being created has graphics. If not, box **516** inquires as to whether the icon is complete. If the icon is not complete, and it has no graphics, then a text icon is being created and box **518** calls the text subroutine shown in FIG. **4**. Upon return from the text subroutine, a pseudonym is created for the completed text icon at box **700**. The creation of pseudonyms at box **700**, as further detailed in FIG. **14**, has already been discussed. The pseudonym for the icon is placed on the icon drum and associated with the rules for that icon at box **520**. Control is then returned by box **522** to box **504** of FIG. **10**.

If, at box **515** of FIG. **11**, the methodology designer indicates that the next icon to be created has graphics, control will pass to box **524** which determines whether the first symbol to be used in creating the icon already exists. If not, the sketch mode is used to construct a new symbol from graphical primitives already in existence (box **526**). Otherwise, the existing symbol is selected at box **528** and placed in the chart work area at box **530**. The rule tool then prompts the methodology designer, at box **532**, to indicate if all connect points for the symbol have been identified. If not, control passes to box **552** which requires the methodology designer to indicate a connect point location. The connector line type

(e.g. line, arc, polyline) and connector line style (e.g. dotted, dashed, solid) are indicated at boxes **554** and **556** respectively. At box **558** the methodology designer states whether directionality has to be indicated. If so, the directionality (e.g. input, output) is indicated at box **560** and control proceeds to box **562** which requires the methodology designer to designate the object that can be connected to the free end of the connector. An object can be a symbol, function, or another schematic diagram. At box **564** the methodology designer states whether additional objects can be connected to the free end of the connector. If so, control returns to box **562** and the process continues until all of the objects that can be connected to the connector free end have been designated. Then control passes to box **566** which determines if all connector annotation has been identified. If not, box **568** calls the text subroutine of FIG. **4**. Once all connector annotation has been identified, control returns, via box **570**, to box **532** of the "create icon" routine.

Once all of the connect points for the icon being created have been identified, box **532** passes control to box **536** for the identification of icon labels. Labels are searchable text fields associated with an icon. In creating labels, the rule tool sets a flag for each label to indicate that it is a searchable item. If labels are to be identified, box **538** calls the text subroutine. Once all labels have been identified, control passes to box **540** to determine if all text in the icon symbol body has been identified. If not, the text subroutine is called at box **542**. Once all symbol body text has been identified, control passes to box **544** where the methodology designer is prompted to indicate whether the icon created is complete. If the icon is not complete, control is returned to box **524** for further building of the icon. Otherwise, a pseudonym is created for the icon at box **700** of FIG. **11**, and the pseudonym is placed on the icon drum and associated with the rules for the icon at box **548**. Control is then returned via box **550** to box **504** of the "create icon drum" routine shown in FIG. **10**.

Turning now to FIG. **10**, once the icon drum is complete, box **504** passes control to box **506** where a map of the complete icon drum is stored as a graphics file. At box **508**, the set of rules to be associated with each icon in the icon drum is stored and linked to the icons. At box **510**, the name of the icon drum just created is added to all of the directories for the function drums that can employ the icon drum. Box **512** then returns control to box **108** of the rule tool flow chart of FIG. **3**.

Once any new icon drums have been created by the methodology designer, the rule tool proceeds to box **110** so that existing icon drums can be modified. If such modifications are to be made, box **111** calls the "modify existing icon drum" routine shown in FIG. **12**.

After entering the "modify existing icon drum" routine at **600**, the methodology designer is prompted at box **602** to furnish the name of the icon drum to be modified. The designated name must, of course, be one for an existing icon drum. Once the icon drum to be modified has been identified, control proceeds to box **604** and the methodology designer is asked whether an existing icon is to be edited. If so, control passes to box **606** and the "edit existing icon" routine shown in FIG. **13** is called.

Turning to FIG. **13**, upon entry to the "edit existing icon" routine at **650**, control passes to box **652** and the methodology designer is instructed to select the icon to be edited and to load the diagram for that icon into the chart work area. Control then proceeds to box **654** and the methodology designer is asked whether it is desired to delete the icon which has been loaded into the chart work area. If so, control passes to box **656** and the icon to be deleted is cleared from the icon drum. At box **658**, the rules associated with the deleted icon are cleared from the set of rules pertaining to the icon drum. Box **660** then returns control to box **604** of the "modify existing icon drum" routine shown in FIG. **12**.

If, at box **654**, the methodology designer indicates that the icon is not to be deleted, then control passes to box **662** which prompts the methodology designer to determine if existing elements of the icon diagram present in the chart work area are to be changed. If so, control passes to box **670** where the methodology designer is required to indicate if prior symbol elements are to be deleted. If deletion of symbol elements is desired, control passes to box **672** and the methodology designer selects the symbol to be deleted from the chart work area. At box **674**, the symbol is deleted, and at box **676** any necessary connector adjustments are made to the remaining portions of the icon. This loop continues until all necessary prior symbol elements have been deleted

Once the methodology designer is finished deleting any prior symbol elements, control passes to box **678** and the methodology designer is prompted to indicate if he intends to move prior symbol elements. If so, control passes to boxes **680**, **682**, and **684** where the symbol elements to be moved are selected, moved, and connectors are adjusted as necessary. This loop will continue until all necessary movement of prior symbol elements has been completed.

After moving any prior symbol elements, control passes to box **686** so that connector adjustments can be made. At box **688**, the symbol for a connector to be adjusted is selected in the chart work area, and at box **690** the connector is deleted or rerouted. When all connectors have been adjusted, control passes to box **692** where text fields are adjusted. The adjustment of text fields is accomplished at box **694** by calling the text subroutine shown in FIG. **4**.

Once necessary text field adjustments have been made, control passes to box **664** where the methodology designer is prompted to determine if more elements are to be added to the icon diagram. Box **664** is also accessed if, at box **662**, the methodology designer indicates that existing elements of the icon diagram are not to be changed. If the methodology designer indicates at box **664** that more elements are to be added to the icon being worked on, then control passes to box **666** and the "create icon" routine shown in FIG. **11** is accessed at box **524**. The "create icon" subroutine has already been discussed. Once any new elements have been added to the icon diagram, the "create icon" routine returns control to box **664**. Then, box **668** returns control to box **604** of FIG. **12**.

Turning back to FIG. **12**, after a methodology designer has attended to any existing icons to be edited, control passes to box **608** where the rule tool prompts the methodology designer to determine if a new icon is to be added. If so, control passes to box **610** and the "create icon" routine shown in FIG. **11** is entered at **514**. After any new icons have been added to the icon drum being modified, control is passed to box **612** and the methodology designer is prompted to indicate if the icon drum being modified is complete. If not, control is

passed back to box **604** and the steps discussed above are repeated.

If the icon drum is complete, control passes from box **612** to box **614** where a map of the modified icon drum is stored as a graphics file. Then, at box **616** the set of rules to be associated with each icon in the modified icon drum is linked with the icon drum and stored. At box **618**, the name of the modified icon drum is added to any additional function drums with which the icon drum can be used, as necessary. At box **620**, control is returned from the "modify existing icon drum" routine to box **110** of FIG. **3**.

After any modification of existing icon drums, control is passed from box **110** in FIG. **3** to box **112** where the methodology designer is asked if any more rule tool work is to be completed. If so, control is returned to box **104** so that additional rule tool work can be accomplished. Otherwise, control is passed to box **114** and the rule tool is exited.

FIG. **15** depicts a set of functions and function primitives which can be supplied with the system to enable a methodology designer to build more complex functions as necessary. These functions are: ADD, DELETE, MOVE, COPY, FIT, SIZE, and REROUTE. Other function primitives, such as the logical "AND", "OR", and "NOT" functions could also be provided.

FIG. **16** depicts an icon drum containing various graphic primitives and icons which might be supplied with a basic system in accordance with the present invention. The graphic primitives and icons suppled will enable a methodology designer to create the more complex icons necessary for use by a problem solving user in generating problem solutions. The graphicI primitives and icons shown are: SOLID LINE, POINT, CIRCLE, ARC, DASHED LINE, DOT/-DASH LINE, DOTTED LINE, DIAMOND, TRIANGLE, RECTANGLE, OR GATE, AND GATE, OVAL, and ELONGATED HEXAGON.

Once a methodology designer has built the set of functions and icons which are necessary to enable a problem solving user to diagram problem solutions for the application in which the problem solving user is interested, schematic drawing of problem solutions can commence. An important feature of the present invention is that during the schematic drawing process, design rule checking can be employed on a continuous basis to assure that the problem solution diagram being generated complies with all rules generated by use of the rule tool. This feature can be accomplished because the rules established for each icon or function are stored in the system, and are always retrieved along with the associated icon or function when the icon or function is selected.

The mechanisms of the schematic drawing process are precisely the same at the level of graphic arts as are those used during the sketch mode of system operation. The use of the system in the sketch mode has already been explained. The difference between the sketch mode and the schematic drawing mode is that in the former, the icons were merely general purpose graphic tokens, whereas in the latter, the icons are formal symbols within an adopted design methodology. Thus, in the schematic drawing mode all of the formal rules and procedures that are attributed to any given step of the methodology, and which can be further attributed to a graphical symbol or token of some form, will be associated with the icon on the display screen of the graphics terminal. In that way, as the problem solving user selects an icon in order to place an instance in the schematic diagram of the solution, the system in parallel, concurrently, and at a very high interactive rate, continuously checks to insure that the design rules associated with the instance of that symbol are adhered to. Thus, the logical structure of any schematic drawing created by the problem solving user as a proposed problem solution must be precisely accurate.

It will now be appreciated that the system of the present invention enables the generation of problem solutions in schematic diagram form wherein the schematic diagrams are inherently methodologically accurate. Once such a schematic diagram has been generated, it can be analyzed, classified, and transformed by other automated systems in a consistent manner to provide desired end results. Among the functions carried out by the system on the formal schematic diagram are the verification and extraction of data to ensure that all required data entry positions and all elements of the schematic diagram are in fact completed. Further, where elements of a design are characterized across several diagrams, and where those diagrams may relate to each other along connecting points thereof, then the references between the diagrams and the contexts imposed by the diagrams can be further verified and checked to insure consistency, and to insure correctness at the level of the referencing information placed in the diagram by a problem solving user. When it has been determined that a schematic diagram is complete, consistent, and internally correct with regard to the parameters and labels employed across several diagrams, then data can be extracted as a means to analyze individual schematics and the set of schematics, as a collection, to provide program description information which is entered into a program description file for subsequent use by other automated systems, such as automatic code generators, integrated circuit mask generators, etc.

It should now be understood that the present invention provides an interactive rule based system for generating problem solutions in schematic diagram form. Schematic diagrams are built from icons (graphic primitives or complex symbols) and functions (which can also comprise icons). Each icon and function to be used in building a schematic diagram representative of a problem solution has certain rules associated therewith. These rules insure that the logical structure of the schematic diagram, as a methodological description, is precisely accurate. By enforcing the rules of use for each icon and function, the system prevents a problem solving user from designing a problem solution which is inconsistent or in violation of the methodology rules. Rule enforcement is accomplished through appropriate prompts by the system during the building of a schematic diagram, and through the use of appropriate error messages when a probelm solving user attempts to take an action which would break a rule.

The system further provides opportunity for a methodology designer to create a library of icons and functions tailored to the specific application in which that methodology designer is interested. Thus, while the system is general purpose, it can be converted by a methodology designer to a dedicated tool for generating problem solutions in a specific field.

An object code listing of the software which implements the sketch mode of the present system is appended hereto as a microfiche appendix. The object code shown is a hexadecimal dump of the executable program code, and can be run on a Wang Laboratories

personal computer (Wang PC) with 640K bytes of memory, a 10 megabyte Winchester disk, and a mouse device manufactured by Display Interface Corporation of Milford, Conn. under the trademark "HiFi Mouse". The Wang PC must include the Microsoft "MS DOS Version 2.01" I operating system and Wang's "BIOS Version 1.21" software.

Although the present invention has been described in connection with a preferred embodiment thereof, many variations and modifications could be made. It is intended to cover all of the variations and modifications which fall within the scope of the present invention, as recited in the following claims.

I claim:

1. A general purpose interactive rule based system for generating problem solutions in schematic diagram form comprising:
- a computer processor;
- a graphics terminal coupled to said processor;
- means for providing a multi-portion split display on said graphics terminal;
- a plurality of functions and graphic primitives stored in said computer processor;
- means for creating a library of icons for an intended application by enabling a methodology designer to:
  - select and arrange said graphic primitives using said graphics terminal, and
  - identify, by way of example, parameters for using each icon in response to prompts initiated by said computer processor;
- means for generating and storing a specific set of rules pertaining to the use of each icon on the basis of the parameters identified;
- means for symbolically displaying in one portion of said split display a set of icons from said library;
- means for displaying in another portion of said split display a set of said functions; and
- means for enabling a problem solving user to access and select displayed icons and functions, and to build a solution to a problem by using functions to graphically couple icons together on a chart work area portion of said split display in accordance with said rules.

2. The system of claim 1 further comprising means for enabling a methodology designer to select and concatenate functions to each other and to icons, using said graphics terminal, to create more complex functions for display and for use in building problem solutions.

3. The system of claim 1 further comprising:
- means for analyzing icons created by a methodology designer to identify open ended connectors having no values assigned thereto; and
- means for prompting a methodology designer to assign input or output values to said open ended connectors;
- whereby said input and output values establish a transfer function across said icon, said icon and values in combination forming a new function for display and for use in building problem solutions.

4. The system of claim 1 wherein said prompts initiated by said computer processor instruct a methodology designer to define points of connection to each icon.

5. The system of claim 4 wherein said prompts instruct a methodology designer to indicate, for each connection point, whether the point is an input, an output, or bidirectional.

6. The system of claim 4 wherein said prompts instruct a methodology designer to indicate for each connection point, the connector line styles which are permitted to be connected to that point.

7. The system of claim 4 wherein said prompts instruct a methodology designer to indicate, for each connection point, the connector line types which are permitted to be connected to that point.

8. The system of claim 4 wherein said prompts instruct a methodology designer to indicate, for each connection point, what other icons or functions are permitted to be connected, through a connector, to that point.

9. The system of claim 4 wherein said prompts instruct a methodology designer to indicate, for each connection point:
- the connector line types which are permitted to be connected to that point;
- the connector line styles which are permitted to be connected to that point;
- whether the point is an input, an output, or bidirectional;
- what other icons or functions are permitted to be connected, through a connector, to that point; and
- whether and what type of annotation is to be associated with a connector connected to that point.

10. The system of claim 1 wherein said prompts initiated by said computer processor instruct a user to identify any fixed and variable text and labels to be associated with each icon.

11. The system of claim 1 further comprising means for verifying that all of said rules are complied with during the building of a problem solution.

12. The system of claim 1 further comprising a mouse input device and a keyboard to enable a methodology designer or problem solving user to interface with the graphics terminal.

13. An interactive system for building schematic diagrams in accordance with formal rules and procedures comprising:
- a computer;
- a graphics terminal coupled to said computer;
- a plurality of general purpose user selectable functions and graphic icons stored in said computer;
- means for storing sets of formal rules pertaining to the use of each general purpose function and icon;
- means for using said graphics terminal to create a library of special purpose icons for an intended application;
- means for generating and storing a set of formal rules pertaining to the use of each special purpose icon in said library of icons;
- means for accessing the set of formal rules for a particular function or icon upon selection of that function or icon by a problem solving user;
- means for enabling a problem solving user to build a schematic diagram by graphically coupling selected icons together on said graphics terminal; and
- means for enforcing the formal rules for each selected icon and function, during the building of a schematic diagram, by prompting a user to input any textual, numerical, or graphic data required by said formal rules and by providing an error message if an action is attempted that would break a rule.

14. The system of claim 13 further comprising:
- means for selecting and concatenating functions to each other and to icons to create more complex

functions for use by a problem solving user in building schematic diagrams.

15. The system of claim 14 further comprising means for:

(a) analyzing icons to identify open ended connectors having no values assigned thereto; and

(b) prompting a methodology designer to assign input or output values to said open ended connectors in accordance with said formal rules,

whereby said input and output values establish a transfer function across said icon, said icon and values in combination forming a new function for use in building schematic diagrams.

16. The system of claim 13 wherein said formal rules for each special purpose icon are generated on the basis of responses to prompting means instructing a methodology designer to define points of connection to each icon.

17. The system of claim 16 wherein said prompting means instructs a methodology designer to indicate, for each connection point, whether the point is an input, an output, or bidirectional.

18. The system of claim 17 wherein said prompting means instructs a methodology designer to indicate, for each connection point, what other icons or functions are permitted to be coupled to that point.

19. The system of claim 18 wherein said prompting means instructs a methodology designer to identify any fixed and variable text and labels to be associated with each icon.

* * * * *

7

911461
2 of 2

An Overview of Logic Synthesis Systems

VINCENT N. TRANS
EXAMINER
ART UNIT 234

Louise Trevillyan

IBM Thomas J. Watson Research Center
Yorktown Heights, New York, 10598

Abstract

The term *logic synthesis* is used to describe systems that range from relatively simple mapping schemes to tools with sophisticated logic optimizations. In this tutorial, the requirements on logic synthesis systems will be discussed and the advantages and disadvantages of different approaches to logic synthesis will be presented.

Introduction

The goal of a logic synthesis system is to increase the productivity of logic designers by automating some or all of the logic design process. In general, this means that such a system automates the production of a technology-specific implementation of a logic function from some more abstract representation of the function. Proposals as to how to accomplish this goal range from straightforward, low-risk mapping schemes to very ambitious, but high-risk, methods of generating logic from high-level algorithms.

In a mapping approach, the design is entered at a low level, either in terms of technology primitives or in terms of "generic" functions which bear a close relationship to the target technology. Very few decisions about the form of the logic need to be made by the system. This method automates some of the more tedious aspects of logic design, and is low risk because the human designer has almost complete control of the resulting implementation, making the success rate for synthesis virtually 100%.

At the other end of the spectrum, there are systems that accept a logic specification in a form similar to a programming language. Almost all of the structural decisions about the logic are made by the synthesis programs. Insertion of memory elements, state assignment, and sharing functional units (ALUs, etc.) are examples of the issues that must be addressed. From the algorithmic description along with constraints on logic size and speed, these systems are expected to produce a complete, usable logic implementation. With current software technology, the likelihood of this being achievable on a system of any size is fairly remote, so the risk of relying on such an approach in a production environment is quite high.

There are also a large number of systems that inhabit the middle ground between these extremes. Most of these programs operate at the *register-transfer* level, in which the designer has made some decisions about the placement of cycle boundaries and the gross structure of the logic, but leaves the decisions about the local structure of the logic up to the program. As would be expected, the success rate of such systems falls between the success rates of the two extremes.

No matter how ambitious the approach, there are certain problems relating to logic quality and usability that must be faced by a synthesis system. Parts I and II of this tutorial describe these problems. In Part III, the approaches to synthesis are categorized. A single system in each category is discussed in order to illustrate how the problems of synthesis are addressed by the approach. The systems chosen are by no means the only such systems. They were chosen because there is enough published information about them to allow evaluation with respect to the criteria given in parts I and II. References [1,2,3] are recommended for additional descriptions and comparisons of recent synthesis systems.

I. Issues in Logic Synthesis

Many synthesis systems deal explicitly with the physical design of the implementation, but in this tutorial, only issues of logic design will be considered. The prerequisite for any synthesis system is that it reliably generate

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1987 ACM 0738-100X/87/0600-0166$00.75

DEF011962

functionally correct, technology-legal implementations. Issues of logic quality are always secondary to those of correctness. Assuming correctness and legality, there are several other criteria by which the quality of synthesis-generated logic should be judged:

Area. This is the classical measure of the effectiveness of a synthesis tool and is usually the first problem addressed by any system. The goal is to make the area, whether measured in terms of cells of a gate-array, devices, equivalent-NAND, or square millimeters of a chip, comparable to that which can be achieved by manual design. Area is not the sole or final measure of the quality of the generated logic. Attention to other constraints, especially timing, can cause the area of the logic to increase. An implementation which meets timing constraints is superior to smaller logic which does not.

Speed. Logic implementations are subject to requirements on the minimum and maximum lengths of paths through the logic. These constraints can be stated in terms of actual arrival and required times expressed in units of time or they may be given in terms of levels of logic. In either case, it is desirable for synthesis to react to timing constraints by making intelligent space-time trade-offs. A good goal for a synthesis program is to generate the smallest logic that meets the constraints on path lengths. The issue of *power consumption* is related to both speed and area, and is usually part of timing correction. Using higher power will tend to speed up logic but will make the logic bigger and hotter. The decision as to the use of power must be balanced with time, space, and total power limit considerations.

Redundancy. A connection is *logically redundant* if it can be replaced by a constant without changing the logic function at any observable point (memory or output) in the design. It is necessary to remove redundant connections both because of chip testability requirements and because excessive connections lead to increased area, path lengths, and wireability problems. The source of redundancies can be the input specification or logic simplifications which cause connections to be moved or logic to be copied.

Physical Design. Although synthesis is not primarily concerned with physical design, some consideration of physical design issues is needed when performing logic design. Even if a synthesized implementation has met all of its logic-level constraints, it still is not a usable design unless it can be placed and wired, and, furthermore, it still satisfies the timing requirements after physical design has been performed.

Additionally, other topics such as error detection/fault isolation, self-test, and simultaneous switching could be addressed by a synthesis system.

II. System Characteristics

The other category of requirements usually has very little to do with the logic design objectives of a synthesis system, but is equally important in terms of a usable tool. No matter how good the quality of the logic, a synthesis system is useless as a design automation tool unless a logic designer can effectively and efficiently use it to help him do his job.

Capacity/Performance. The main justification for a synthesis system is usually that, with the advent of VLSI, the complexity and amounts of logic that need to be designed have increased so greatly that manual design is ineffective or too expensive. A useful synthesis system should have algorithms that scale in a reasonable way with the size of the design, and should either be able to handle very large flat designs or should provide hierarchical capabilities. If hierarchy is used, it is desirable to have some mechanism for cross-partition simplifications to handle global issues (such as redundancy removal).

Stability. We will call a system stable if a "small" change in the input specification produces a correspondingly small change in the output implementation. This is a useful attribute in cases where the designer is attempting to "tune" the design. For example, a designer loses productivity when a change to the specification to improve the timing on one path causes another path to become too long or an explosion in area in some other part of the logic.

Flexibility. The system should be able to accommodate varying input formats, technologies, design styles, and project requirements.

Output interfaces. The outputs of the synthesis system should not require manual manipulation to prepare it for further processing, and its output should be complete in the sense that everything needed for further processing is provided by the synthesis system. In addition to the design itself, this might include separate files of timing or verification data, graphic forms of the logic, or a form suitable for entry into a logic editor.

Interactive capabilities. It may be useful to allow the designer to "guide" the synthesis process by applying simplifications to only some parts of the logic or by varying the order of optimizations. One of the arguments in favor of synthesis is that the output is "correct by construction". If interactive systems include the capability of editing the logic, this advantage may be lost.

DEF011963

Usability. The system should be convenient for a logic designer to use. This means that consideration should be given to topics such as the design of the input language or editor, the amount of training a designer needs to use the system, and the understandability of the output of the system. Judging usability requires "hands on" use of a system. Since this was not possible, no attempt will be made to evaluate the usability of the systems discussed in this tutorial.

Other important topics related to synthesis are *logic comparison* and *incremental processing*. The goal of logic comparison is to verify that two logic models perform the same function. It can be used as a check on a synthesis system, when manual changes are made to synthesized logic, or in doing incremental processing. Incremental processing is most useful when employed together with an incremental physical design system. Given the large size of today's chips, physical design can be very time-consuming. When the logic is modified after physical design has been performed, it is useful if the synthesis system can bound the logic changes so that incremental physical design can be done. This can be done either by comparison of the original synthesized design with the new one and generating a change file, or by using hierarchical techniques in synthesis that bound the change to the size of the piece that has been modified. In the latter case, the responsibility is on the designer to provide suitably sized partitions in the hierarchy.

III. Approaches to the Problem

As has been mentioned, the range of synthesis systems, in terms of the problems they address, the sophistication of the algorithms, and the level of ambition of the systems, is very broad. In this section, we will examine five existing synthesis systems that typify the various approaches.

Polaris, a mapping system

One of the most simple, but extremely effective, synthesis systems is the one developed by Hitachi and used on virtually all of the chips in its M-68X processor [4,5]. The objective of this system is to always give the designer exactly the logic he expects with respect to gate counts and path lengths and to satisfy the electrical and physical constraints of the technology. The design is entered at a fairly low level in the ALDL language and the synthesis algorithm "maps functional specifications to target technology components on a one-to-one basis" [5 p 367]. The system does in fact make changes to the logic. It coalesces identical nodes and performs fan-in adjustment while translating the language into a graph form. It follows this by mapping the logical gate structure into allowed physical gates using a levelized polarity-propagation scheme. The system contains a table of physical gates that is ordered by desirability of use. The polarity propagation proceeds from the logic outputs toward the inputs, choosing the highest priority physical gate that includes the correct logic function and satisfies physical requirements (such as fan-in and dotting rules). The choice of the function to be used at a gate determines the required polarities of the logic that feeds the gate. Inverters are inserted when there is a polarity mismatch. Finally, a table-driven process is used to map the logic from physical gates into the physical units which correspond to the primitives of the gate-array.

In addition to synthesizing logic, the system is able to use complex modules specified by the designer and customize them to their place of use by deleting unused logic terminals. It can also copy hand-designed "basic modules" into the logic.

The Hitachi system addresses problems of correctness and legality but leaves issues of area, speed, etc. largely in the hands of the logic designer. The algorithms used appear to be linear in the design size, and, because it makes no dramatic changes to the logic, it rates high on stability and flexibility. It is not clear how many gates the system can actually handle, but the functional units used for synthesis appear to be in the 30-50 gate range. Logic comparison and incremental processing capabilities are available [6]. The small size of the models probably has more to do with the incremental synthesis and physical design capabilities than with any restriction on the synthesis system itself.

Productivity is gained in the Hitachi framework by relieving the designer of the necessity of doing many of the tedious and mechanical jobs required in gate-array/standard-cell design. However, most of the detailed logic decisions are still left to the designer to handle. Other systems which also operate at a register-transfer level are more ambitious in this respect. The large number of "higher-level" register-transfer level synthesis schemes can be classified roughly into three paradigms: algebraic, compiler-like, and expert-system. In the following sections, we will focus on an example of each of these approaches.

APLAS, an algebraic approach

Algebraic methods for automated synthesis generally are based on the approach pioneered by Quine [7], McCluskey [8], and others the 1950s and are rely on the minimization of *programmable logic arrays*, two-level AND-OR representations of the logic. (While most of the algebraic systems are based on PLAs, there are some that have taken a different approach, most nota-

DEF011964

bly the MACDAS system [9].) Much of the work done in this area has to do with *PLA minimization*, the goal of which is to reduce the number of AND terms in the implementation. There are known algorithms which will always lead to a minimal implementation (in terms of unrestricted AND and OR gates). These have very large computational requirements so that a medium-sized PLA with about 20 inputs is the maximum unit of logic that can be processed using these methods. The idea behind recent work has been to find new approaches, often employing heuristics, that will achieve or approach minimal solutions but avoid the long run times associated with the classical methods.

The APLAS system [10], developed at Stanford University, is an example of a PLA-based algebraic synthesis approach. The goal of the system is to develop a practical synthesis method that will implement the control logic portion of a design as PLAs.

The input to APLAS is a register-transfer level description done in DDL-P. The language is translated and produces a list of Boolean equations in terms of the register and logic inputs and outputs. The next process reads in the equations, converts them to an AND-OR form, and uses PLA techniques to to minimize the number of product terms needed to implement the function. In this case, heuristics are used in the minimization procedure, so an optimal solution is not guaranteed. In practice, the solutions are very good, and, for small examples, optimality is achieved.

The minimized form is further reduced by a procedure to optimize the number of connections in the PLA, and then the large PLA is partitioned into smaller PLAs in order to reduce area and improve speed. The partitioner also contains a redundancy removal algorithm.

This system addresses the major issues of synthesis. Since one of the advantages of PLA design is that it is very each to place and wire PLAs, this system also has good characteristics with respect to physical design. Programs that handle a design as a single PLA are notoriously poor with respect to capacity/performance, and this one is no exception (the maximum model reported has 72 inputs). The only design style supported by APLAS is PLA output and it would be difficult to change to a different design style. This is an inherent problem with this approach, but algebraic methods can be combined with other types of systems in ways the enhance their flexibility, as will be described later.

LSS, a compiler-like system

Another approach to the problem of synthesis has its roots in the observation that optimizing logic is similar to optimizing programs. The compiler-like synthesis systems tend therefore to draw on the work done in optimizing programming-language compilers. Compiler optimizations, such as common subexpression elimination, constant propagation, code motion, dead code elimination, and procedure integration, are re-cast into forms that are useful in logic optimizations. For example, in programming-language compilers, code motion might be used to move computations out of loops; in synthesis systems, the same techniques might be used to move late signals closer to the registers and outputs of the logic.

The synthesis system from IBM, LSS [11], is an example of a system built using compiler technology. IBM uses many different register-transfer level languages, and most of these can be used as input to LSS. The input is translated to a dataflow graph in which the nodes represent operations specified in the input (DECODEs, parities, adders, etc. as well as ANDs, ORs, and NOTs). The graph edges represent the data relationships in the logic. Peephole optimizations in the form of local, pattern matching transformations are applied at this level. The logic is then translated to a NAND or NOR level where global transformations similar to dataflow-based algorithms in programming-language compilers are used to simplify the logic [12] and redundancy removal processing [13] is applied. The early optimizations are largely directed at area minimization, with timing a secondary consideration. In the final phases, the logic is mapped into the target technology and detail timing analysis and correction are done. At this level, the focus is timing and power consumption with area being a secondary factor. These are generally accomplished by a global-analysis/local-correction approach.

Specific technology information in the form of tables and "user exits" is used throughout the synthesis process to aid in making optimization decisions. Externalizing such data allows a trained person to easily adapt LSS to new technologies and user requirements. Because global algorithms are being used, there is no guarantee of stable output - a small change to the source could potentially change a large window of logic.

LSS contains simplifications intended to address the major issues of logic synthesis. Like programming-language compilers, it is a "turnkey" system and the user interacts with it only at the source-code level. With the exception of full redundancy removal, the algorithms scale in a roughly linear fashion with the model size. (Complete redundancy removal is optional and used only when there are severe testability problems with the logic.) Models containing 40,000 two-way NAND equivalents have been processed using LSS. Logic comparison is done by using the SAS system [14]. LSS is used widely within IBM and hundreds of production bipolar and FET chips have been produced.

DEF011965

SOCRATES, an expert system

The major observation behind the expert-system approach to synthesis is that the task of synthesis, with its multiple, conflicting objectives, is a very difficult problem to solve algorithmically. The expert-system approach overcomes this problem by encoding the expertise of logic designers into a set of rules and allowing a system to apply the rules in varying sequences in such a way as to drive down the cost of the logic implementation. The significance of the arbitrary order of the application of the rules is that the search space of implementation possibilities is increased and the local minima typical of other approaches are avoided.

The SOCRATES system [15,16,17] General Electric Microelectronics Center and the University of Colorado at Boulder is a particularly interesting example of this kind of system because it illustrates not only an expert system, but a way in which an algebraic approach can be used in conjunction with another method to produce good technology-dependent, multi-level implementations.

SOCRATES can accept input in the form of Boolean equations, "linked" single-output PLAs, or a net-list. No matter what the input form, the logic is translated into a directed acyclic graph in which each node represents a two-level Boolean equation. Minimization is carried out at this level by using the ESPRESSO IIC logic minimizer [18]. This simplification is followed by another algebraic synthesis step which includes *weak division* (a form of factoring), and *multi-level minimization* (to take advantage of don't-care states). At this point, the algebraic portion of the system has designed the best technology-independent implementation it can. A library-mapping phase is then run to translate the function into a circuit in terms of a small number of generic primitives and the rules-driven portion of the system is run to complete the optimization.

The rules-based approach is used to take advantage of special features of the target technology. Each rule describes a local transformation that would, if applied, change a small window of logic so as to reduce the cost of the circuit in terms of area or performance. The tasks that the system performs are: (1) determine the set of rules which apply to the circuit, (2) compute the cost function for each application to determine the goodness of the moves, (3) select the move to be made, and (4) apply the selected rule to the logic and update the cost function.

The quality of the resultant implementation depends on the rules and the order in which they are applied. Because the rules interact, it is desirable to be able to look ahead through a series of rule applications when selecting the rules to apply. The rule ordering can be viewed as a tree in which the breadth represents the number of rules that can apply at a given place and the depth represents the sequence of transformations that can be done. In the ideal case, the complete tree of rule applications would be searched to find the optimal implementation (with respect to the rules). In practice, this is too expensive so the search must be limited to a certain breadth and depth of the tree. In SOCRATES, there is a second rule-based system, called the *meta-system*, that controls the search strategy and dynamically adjusts the scope of search in the tree.

SOCRATES addresses the problems of area and speed by accounting for these things in its cost function. It is less clear how the problem of redundancy is solved, for even if a completely testable design could be produced from the algebraic portion of the system, subsequent modifications to the logic cause logical redundancies to be introduced. Since redundancy is a global phenomenon, a local approach cannot deal effectively with the problem.

The flexibility of the system is enhanced by a "rule generation" module which allows a sophisticated user to enter and test a new rule in about three minutes. The ability to easily add such rules means that the system can be readily customized to new design styles, technologies, or project requirements. Again because of global algebraic algorithms, there is no guarantee of stability in this system.

Performance is a problem for most expert systems. In SOCRATES, even though great care has been taken in the system design and in efficient searching strategies, the results given in [15] do not indicate that the run time scales well with the size of the final design. The largest model the authors have reported running is less than 400 two-way NAND equivalents, but there is no reason to believe that this is anywhere near the maximum capacity of the system.

Flamel, a high-level system

The approaches discussed above all use register-transfer level input. The final area of synthesis attempts to raise the level of specification to an algorithmic level, so that the input would look more like a PASCAL or an ALGOL program. While such systems still fall into taxonomy given above, they are extended to make major structural decisions about the form of the logic. The system that is discussed here is compiler-like, but expert systems are also being used, most notably at Carnegie-Mellon University [19].

Flamel [20], developed at Stanford University, uses as input a program written in a slightly restricted version of PASCAL and execution frequency counts from a typical run of the program. Its goal is to find a hardware design with the same "external behavior" as the program which minimizes the estimated run time on the

DEF011966

typical data and which honors users' constraints on the cost of the implementation.

The program is translated into a control-flow graph in which the nodes represent the *basic blocks* of the program and the edges represent the flow-of-control of the program. Each basic block (a single-entry-single-exit section of code from the program) is represented by a dataflow graph together with some sequencing information. The program applies transformations to the control-flow graph to attempt to merge basic blocks in ways that can be used to enhance parallelism and reduce time. The choice of transformations to be applied is controlled by generating a tree of transformation combinations and picking the set of nodes in the tree that minimize the estimated time while adhering to a user-supplied constraint on resources (number of ALUs, registers, etc.). The time estimate is based on the number of levels in the block and on the frequency of use of the block as given in the execution frequency input. This process is followed by one that shortens the distance along critical paths by applying a level compression algorithm. Level compression also yields further opportunity for parallelism. Finally, Flamel optimizes resources by "folding" together pairs of resources that perform (or could be made to perform) the same function and are not used simultaneously. In some cases, the schedule will be lengthened to form more opportunities for folding in order to meet resource constraints.

Flamel's output is a description of the data path consisting of the ordered functional units (ALUs, adders, I/O pads, etc.) and the busses and their attachments, and a description of the control logic in terms of the states and transitions of a finite-state machine. All of the structural decisions have been made to get a design that meets the user's constraints. At this point, other tools, such as the register-transfer level systems described above, could be used to finish the task of detail logic design.

The results given for Flamel indicate that it is very responsive to the constraints supplied by the user and that it can quickly generate implementations from the high-level description. At this point, the resulting implementations are much larger than can be done manually; on the other hand, it can produce acceptable designs in a small fraction of the time required to do a manual design.

Summary

In order to be successful at logic synthesis, a system must do more than simply minimize area. It must generate implementations that are acceptable in terms of other measures such as speed, testability, physical design, and power. Furthermore, if it is to be useful in a VLSI environment, it must pay attention to the system characteristics of capacity, performance, stability, flexibility, and usability.

There are four basic approaches to the synthesis: mapping, algebraic, compiler-like, and expert-system. Each of them has been shown in real implementations to be viable, and each method has strengths and weaknesses in terms of the quality of logic that is generated and in the characteristics of the approach.

Since the goal of logic synthesis is to increase the productivity of the human logic designer, an evaluation of the success of the method should take into account the risk associated with using the system as well as the amount the system can do for the designer. A mapping approach may not be ambitious but it is very safe and provides a great deal of automation. On the other end of the spectrum, the "synthesis from algorithms" approach is the goal that everyone would like to reach, but it is not really practical yet in a production environment.

Acknowledgements

I would like to thank Bill Joyner, Terri Nix, and Dan Ostapko for their suggestions and help in developing this paper.

References

[1] W.H. Joyner, Jr., "Logic Synthesis", *Proceedings of Comp-Euro '87*, Hamburg, Germany, 1987.

[2] A.R. Newton, "Techniques for Logic Synthesis", 1985.

[3] A.L. Sangiovanni-Vincentelli, "An Overview of Synthesis Systems", *IEEE Custom Integrated Circuits Conference*, Portland, Oregon, 1985, pp. 221-225.

[4] T. Shinsha, et. al., "POLARIS: Polarity Propagation Algorithms for Cominations Logic Synthesis", *Twenty-first Design Automation Conference*, Las Vegas, NV, 1984, pp. 221-225.

[5] Y. Tsuchiya, et. al., "Establishment of Higher Level Logic Design for Very Large Scale Computers", *Twenty-third Design Automation Conference*, Las Vegas, NV, 1986, pp. 366-371.

[6] T. Shinsha, et. al. "Incremental Logic Synthesis Through Gate Logic Structure Identification", *Twenty-third Design Automation Conference*, Las Vegas, NV, 1986, pp. 366-371.

[7] W.V. Quine, "The Problem of Simplifying Truth Functions", *Am. Math. Monthly*, Fall, 1952.

[8] E.J. McCluskey, Jr., "Minimization of Boolean Functions", *Bell System Tech. Jour.*, Vol. 35, No. 6, November, 1956.

[9] T. Sasao, "MACDAS: Multi-level AND-OR circuit synthesis using two-variable function generators", *Twenty-third Design Automation Conference*, Las Vegas, NV, 1986, pp. 86-93.

[10] S. Kang and W.M. vanCleemput, "Automatic PLA Synthesis from a DDL-P Description", *Eighteenth Design Automation Conference*, 1981, pp. 391-397.

[11] W.H. Joyner, et.al., "Technology Adaptation in Logic Synthesis", *Twenty-third Design Automation Conference*, Las Vegas, NV, 1986, pp. 94-100.

[12] L. Trevillyan, W.H. Joyner, C.L. Berman, "Global Flow Analysis in Automatic Logic Design", *IEEE Transactions on Computers*, vol. C-35, no. 1, January, 1986.

[13] D. Brand, "Redundancy and Don't Cares in Logic Synthesis", *IEEE Transactions on Computers*, vol. CAD-5, no. 10, October, 1983, pp. 947-952.

[14] G. L. Smith, R.J. Bahnsen, H. Halliwell, "Boolean Comparison of Hardware and Flowcharts", *IBM Journal of Research and Development*, vol. 26, no. 1, January 1982, pp. 106-116.

[15] A.J. deGeus and W. Cohen, "A Rule-Based System for Optimizing Combinational Logic", *IEEE Design and Test of Computers*, vol. 2, no. 4, August, 1986, pp. 22-32.

[16] K. Bartlett, W. Cohen, A. deGeus, G. Hachtel, "Synthesis and Optimization of Multilevel Logic under Timing Constraints", *IEEE Transactions on Computer Aided Design*, vol. CAD-5, no. 4, October, 1986, pp. 582-596.

[17] W. Cohen, K. Bartlett, A.J. DeGeus, "Impact of metarules in a rule based expert system for gate level optimizations", *Proc. IEEE Int. Symp. on Circuits and Systems*, June 1985, pp 873-876.

[18] R.K. Brayton, et. al., *Logic Minimization Algorithms for VLSI Synthesis*, October, 1986, pp. 582-596. Hingham, MA: Kluwer Academic Publishers, 1984.

[19] T.J. Kowalski, D.J. Geiger, W.H. Wolf, W. Fichtner, "The VLSI design automation assistant: From algorithms to silicon", *IEEE Design and Test of Computers*, vol. 2, no. 4, August, 1986, pp. 33-43.

[20] H. Trickey, "Flamel: A High-Level Hardware Compiler", *IEEE Transactions on Computer Aided Design*, vol. CAD-6, no. 2, March, 1987, pp. 259-269.

DEF011968